DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

1

AFIT/GCS/ENG/88D-2

"Original contains color plates: All DTIC reproductions will be in black and white"

Volumetric Rendering Techniques for the Display of Three-Dimensional Aerodynamic Flow Field Data

THESIS

David James Bridges
Captain, USAF

AFIT/GCS/ENG/88D-2

Approved for public release; distribution unlimited

AFIT/GCS/ENG/88D-2

VOLUMETRIC RENDERING TECHNIQUES FOR THE DISPLAY OF THREE-DIMENSIONAL AERODYNAMIC FLOW FIELD DATA

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

David James Bridges, B.S.
Captain, USAF

December, 1988

Acknowledgments

I've had a great deal of help from a great many people during my thesis effort. Some of these people were with me constantly; others have never met me. I'd like to thank them all.

First, I'd like to thank my thesis advisor, Maj Phil Amburn for helping me throughout this effort. Maj Amburn has always been there to encourage me when things were going wrong. The final results of this thesis would have been substantially diminished without his guidance.

I'd also like to thank Capt Randy Jost and Capt Phil Beran for explaining flow fields, providing data files, and editing my thesis text. I hope they feel like they have gotten something in return for their efforts.

I especially appreciate the faith shown by my sponsors, Dr Larry Rapaganni and Mr Bob Conley of the Air Force Weapons Laboratory, Kirtland AFB, NM. While my program hasn't been tested on their equipment yet, I hope it will be useful to them.

Special thanks go to Mr Marc Levoy for providing an advance copy of his article "Display of Surfaces from Volume Data". His work was exactly what I needed and was presented exactly when I needed it.

The people to whom I owe the greatest thanks are my family. They've put up with my late night sessions on the computer, weekends when I couldn't afford any family activities, and eighteen months when my body was at home but my mind wasn't. This thesis is dedicated to [redacted] Thank-you for everything that you've given to me.

David James Bridges

Table of Contents

List of Figures

List of Tables

Abstract

The behavior of fluids has been studied for years. This behavior has been modeled by the Navier-Stokes equations since the mid-nineteenth century; however, these equations are so complex that their use by engineers was impractical until the advent of the modern computer. The same computers that made these equations useful created a new problem: data saturation. The solution to the Navier-Stokes equations which represent an aerodynamic flow field is comprised of thousands if not millions of numbers. This much data makes it virtually impossible to conceptualize the situation within the flow field. To alleviate this problem, various computer graphics techniques have been used to provide images of flow fields. Some of these techniques rely on approximating isometric surfaces of interest with polygons; others only provide an image of a two-dimensional "slice" of the flow field.

This thesis project applies a relatively new graphics technique known as "volumetric rendering" to generate three-dimensional images of aerodynamic flow fields. Volumetric rendering has shown promising results in several applications of scientific visualization. One advantage of this technique is that it doesn't rely on geometric primitives to approximate a surface. Also, no *a priori* knowledge of the flow field's form is required to produce the final image.

To assess the utility of volumetric rendering in the field of computational fluid dynamics (CFD), a program known as VIPER is developed. This thesis outlines the requirements and specific algorithms for this program. Images of flow fields are presented and discussed along with program enhancements. Although VIPER is developed for CFD purposes, its design permits its use in different applications, several of which are discussed.

VOLUMETRIC RENDERING TECHNIQUES FOR THE DISPLAY OF THREE-DIMENSIONAL AERODYNAMIC FLOW FIELD DATA

I. Introduction

1.1 Thesis Statement

A relatively new computer graphics technique known as volumetric rendering is being recognized as a powerful means of scientific visualization in such diverse fields as meteorology, medicine, and molecular physics. This thesis will show that volumetric rendering should also be developed as an engineering tool for producing detailed images of theoretically- and experimentally-derived three-dimensional (3D) aerodynamic flow fields. By providing a 3D image of this data, the time required to test and perfect aerodynamic designs may be significantly reduced, thus increasing engineering efficiency and productivity.

1.2 Background

Flow field visualization is an essential tool in many fields of engineering. Whether a design is for a building that must withstand high winds, a ship that must negotiate heavy seas, or a supersonic fighter that must pass through the air, the engineer must be able to study how the structure interacts with its fluid environment. Ever since engineers started working with fluid dynamics problems, they have been faced with the same problem: how to visualize the invisible. Until now, engineers have relied on two basic approaches to this problem. The first of these approaches is to build a physical model of the object under design and actually observe how the object behaves in a flow field. This technique typically relies on special test equipment such as wind tunnels. The second, more recent approach is to use a computer to solve the Navier-Stokes equations for the object and render a two-dimensional (2D) image based on these solutions. Both of these approaches have severe limitations.

The Wright brothers were pioneers in the development of experimental flow fields. Their first development in this area was a bicycle-mounted device which held model airfoils in the flow field that developed as the bicycle was pedaled through town. Although useful, this arrangement was too susceptible to uncontrolled conditions (such as variable winds) and inaccurate measurements (due to operator overload) to be used for detailed engineering work. To overcome these shortcomings the Wrights invented the wind tunnel, a device that could generate a constant, controlled flow field wherein the interaction between airfoils and their fluid environment could be carefully measured (1:151). Since then, engineers have relied extensively on flow field studies performed in wind tunnels. Although the wind tunnel has proven itself an invaluable tool to the engineer, it requires cumbersome and expensive procedures (1:1).

To begin with, a physical model of the object under test (OUT) must be constructed. These models range in size from under an inch in length to full-size model aircraft. If the object's design needs to be changed, the model must also be be modified or completely rebuilt. When the model is finished, it must be prepared for testing. This process requires mounting the model in a wind tunnel and preparing the instrumentation required for the test. Once the model and wind tunnel are ready, the test is run. Finally, after the test is completed, the instrumentation and model must be removed from the tunnel. If more data is required later, the entire test sequence must be repeated. Every step of this process involves time and money; in an era of tightened research budgets, this cycle is too expensive to be performed often.

Another constraint placed upon the engineer is the availability of adequate wind tunnels. There are relatively few wind tunnels available to engineers. If there is no wind tunnel available, an engineer's project must be delayed or possibly canceled altogether.

Fortunately, the recent increase in computer capabilities and availability have presented solutions to many of these problems. To begin with, the computer uses a model that is abstract rather than physical. When compared to physical models, an abstract model is relatively easy to construct and modify. Secondly, computers are now commonplace and available to virtually every engineer. Finally, no specialized instrumentation is needed to run a flow field test on a computer; only a computer with sufficient memory and some means of displaying the test results is required. However, the same computers that provide this abstract design capability also create a new problem: data saturation. Therefore, computer availability is not enough; a satisfactory method for displaying and analyzing test results still poses a serious problem to researchers.

Flow field data is normally represented by a three-dimensional (3D) mesh of data points. These data points are composed of solutions to Navier-Stokes equations (a set of mathematical equations that model fluid behavior). Each solution may contain values for as many as twenty variables. If a flow field is modeled by a 100x100x100 node mesh there will be 1,000,000 locations where data is to be analyzed. This volume of data is too much for a engineer to assimilate from printed numbers alone. Therefore, a graphic representation is necessary.

Early efforts at flow field visualization were based on modeling the action of wind tunnels themselves. One such system, called CINEMA, was designed by Capt E. P. Amburn and used a technique known as "particle tracing". This system simulated the release of particles in a flow field and traced the particles' paths through the field as a function of elapsed time (see Figure 1). Local velocities at each node were stored in a table. As a particle neared a node, its location and velocity were updated by referencing the node table (1:21-25). CINEMA produced 2D and 3D images and used vector-based graphics systems (1:16-17). This system was an improvement over contemporary systems in that it displayed temporal features. However, the use of a vector-based system precluded a continuous display between data points. Thus, if the user released particles in the wrong areas, critical features could be omitted from the final flow field image (*Note:* Vector-based displays have the advantage of clean, crisp lines and curves; however, it is difficult to fill in solid areas (patches) of color due to hardware timing constraints. Raster-based graphics displays are better suited for patch-filling. A full discussion of vector- and raster-based graphics displays may be found in Chapter 3 of (7).). In spite of this shortcoming, particle tracing is still one of the more popular techniques used for rendering flow field images. It is a relatively fast technique and lends itself nicely to user interaction. GRAPH3D, for example, is a software package developed for NASA's Ames Research Center. Although several different methods for rendering images of flow fields are provided, the particle tracing method is still considered the most useful technique of those available on GRAPH3D (2).

In 1985, Kenneth Kroos presented a study of 3D visualization techniques involving vector-based displays. Kroos' efforts involved displaying various shapes at each data point to show trends within a flow field. The primary objective of these experiments was to develop an algorithm that could quickly generate these images (11:130). Kroos concluded that the optimum graphic symbol for this purpose was a simple arrow-shaped vector. Such a vector is "anchored" to each data point and has its magnitude and direction determined by the fluid flow at its origin. Although this technique allows fast rendering of an image, Kroos notes that it has several drawbacks (11:111). First, due to the relatively small size of most graphics devices it is difficult to display very large and very small vectors in the same image. Another problem posed by this technique is its inability to reflect temporal features; thus multiple images must be produced to show time-dependent features

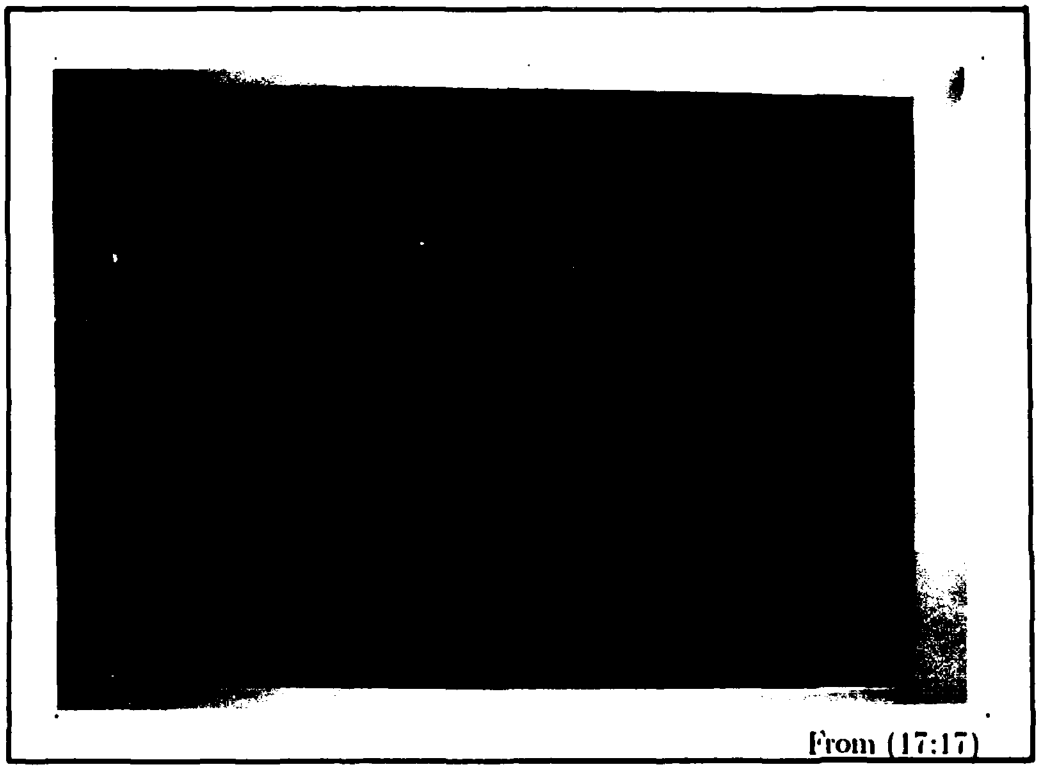

Figure 1. Typical Particle Tracing Image

within the flow field. Also, when rendering an image of a large flow field, this technique produces a very complex display that makes it difficult to interpret the image. Finally, this technique requires the viewer to "fill in the blanks" to interpolate flow field values between data points.

Winkelmann and Tsao were among the first to develop a raster-based system for flow field data display. Their approach took advantage of a raster-based system's ability to fill in and color solid areas on the display. This capability allowed them to show full internodal value interpolations and provide qualitative information by associating specific colors with different parameter values. Winkelmann and Tsao demonstrated the accuracy of their system by comparing computer-generated images of flow field data with photographs from wind tunnel tests; a very close correspondence between the two images indicated the validity of this technique (21:177). However, their system was limited to 2D displays.

Recognizing the limitations of using 2D images, Smith, Sperry, and Everton produced a technique that attempts to use images similar to Winkelmann and Tsao's to build a 3D image. Their approach generates "slices" of flow field imagery, then reorients and displays them in their positions relative to the flow field, as shown in Figure 2 (22:197). Although this approach is more useful than the "flat" images produced by Winkelmann and Tsao, it still leaves gaps in the image, thus forcing the user to guess what the flow field looks like within these voids

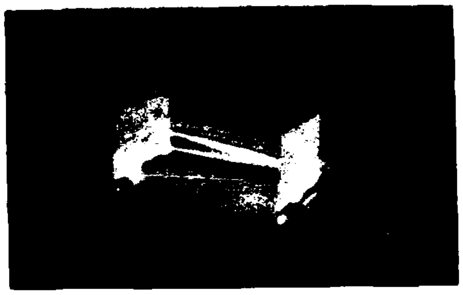
From (17:12)

Figure 2. Three Dimension Image Formed By Several 2D Images.

A completely different approach to rendering data was developed by William Lorensen and Harvey Cline for rendering medical data. Their "Marching Cubes" algorithm (16) assumes the data being used represents a volume comprised of cubic volumes ("voxels"). Each voxel has eight vertices; a specific data value is associated with each vertex. Once the volume's orientation relative to the user has been established, the rendering process begins. Each row of voxels is examined (from front to back) for user specified features of interest. In order to determine the existence of these values, a vertex is considered a "0" or a "1", depending on whether its data value is below or above the value being sought. Once all eight vertices have been processed, a surface is developed. This is accomplished by performing a table lookup based upon the pattern formed by the vertex classifications. This lookup will reduce the surface to one of fourteen possible surfaces that can be produced by linking the voxel's edge midpoints. After assigning a directional normal based on the surface's gradient and orientation, a lighting model is applied, the surface is rendered, and processing moves on to the next voxel (16:164 - 166). This algorithm represents a radically new technique known as volumetric rendering. Its advantage is that the data used is a 3D set of values rather than a collection of planar polygons. However, the marching cubes algorithm operates by assuming that any intersecting planes will pass through either the edge midpoints or the vertices of any given voxel. Based on this assumption, the algorithm then reverts to using planar polygons to produce a surface of interest.

One problem encountered with 3D techniques is that they have traditionally tried to fit geometric primitives such as triangles and quadrilaterals to the surface of an object (3:109-120). This technique provides the advantage of well-known methods for deriving essential surface information. However, there are several drawbacks to this approach when attempting to render a flow field image.

To begin with, using this technique can be difficult when dealing with complex surfaces; "holes" can appear in the surface being rendered due to tridirectional polygon fitting. This problem is analagous to fitting square pegs into round holes; when a polygon is used to build a smooth surface such as a sphere, wrinkles and gaps may appear. Wherever these problems occur, special interpolation routines need to be invoked in an effort to "patch" the surface (19:170). This interpolation may lead to an inaccurate image.

Another drawback to polygon fitting techniques is the restriction they place on the viewing of underlying information. Unless time-intensive sorting routines are used, only the outermost shell of data can be displayed and background details are masked. A final drawback to these techniques is that they tend to be based on binary decisions. Thus, a portion of the field either is or isn't contained within the surface; no allowances are made for "fuzzy zones" where portions of the field are approximately (but not exactly) equal to the isovalue selected for rendering. This shortcoming may mask critical shear areas and deprive the engineer of crucial information. Herein lie the weaknesses of the marching cubes algorithm: (1) the desired data value(s) will not necessarily pass through the edge midpoints, and (2) curved surfaces may only be *approximated* by triangles. Furthermore, although surfaces are common in medical images, the only real surfaces within a flow field are those of the object being tested. The flow field itself has no surfaces *per se*; it is a volume of scalar values. However, I believe that volumetric rendering techniques *may be used to produce accurate and useful images* of 3D aerodynamic flow field data by developing surfaces of isometric values.

Robert Haber has stated that "*Volumetric rendering* methods are an important new development for visualizing three-dimensional systems (12:95)." Marc Levoy has developed a volumetric rendering technique for medical and electron density visualization (15:29). This technique doesn't rely on geometric primitives; instead, the volume of data is rendered by using ray-tracing techniques (as described in (10) and (23)). Simulated light rays are fired through the array of voxels. As a ray passes through a voxel, the data value at the point of intersection between the ray and the voxel is trilinearly interpolated from the data values at the voxel's eight vertices. Based on this value, the color and transluscence of the voxel is determined and factored into the ray's final color. When the ray has passed through the entire volume, the color of the ray is transferred to the pixel through which the ray passes. When all pixels have been colored, the image is complete.

This approach has several advantages. Because no geometric primitives are being forced into a surface, no gaps will be generated. The absence of gaps (or patches covering gaps) means that an image's accuracy is limited only by the resolution of the display device being used and the density of the data mesh. Secondly, the transluscence of a voxel is determined by its value's proximity to a preselected target value; areas where data values are "close" to the target value will be "slightly" colored (15:32). This color gradation gives the researcher visual clues about the gradient and thickness of an isovalue surface. Object transluscence also permits a view of underlying and background structures. As Levoy notes, his technique may be "easily modified to render interstitial volumes as semi-transparent gels (15:36)". Based on Levoy's results and the fact that the electron density data set he used is similar to a flow field data set, I believe that this technique can be adapted to flow field visualization to produce 3D images of an entire aerodynamic flow field.

1.3 Basic Requirements

To be a fully effective engineering tool, an image generator should have the following capabilities:

a. Data should be displayed throughout the entire flow field image rather than just in isolated areas. This capability provides a complete image, thus assuring the engineer that potentially critical features are not being omitted. Conversely, the user should be able to isolate specific subvolumes in order to eliminate regions of little or no interest.

b. The object being tested should be included in the image along with the flow field data. The display of critical flow field features is of little value if there is no way to correlate these features with their origin or how they affect the OUT.

c. Parameter values to be displayed should be user controlled. Many values will be present within a flow field. If all values are displayed, the image will be confusing. If too few are displayed, critical data may be omitted. By allowing the user to select target values for selected parameters, the system can render images that isolate specific values of interest.

d. The range of acceptable deviation from a target parameter value must be user-controlled. This feature allows the user to focus the imaging system as finely as he wishes.

e. Different colors should be associated with each target value selected by the user. Without color differentiation, there is no way to distinguish one target value from another. A color key should also be provided in the image. This key should identify the value associated with each color in the display.

f. Relationships between parameters should be user-controlled. There are five basic flow field parameters: pressure, energy, and three flow velocity vector values. All other flow field values may be derived from these parameters. However, the user must be able to specify how these parameters are interrelated before the values can be generated and the results displayed.

g. The system must allow the user to specify the viewing geometry. This involves positioning the viewpoint, the viewing plane, the volume being rendered, and all light sources.

h. Temporal features should be displayed. Many flow field features (such as vortices) develop and dissipate over a period of time. In order to study these features, their entire lifespan must be rendered. Ideally, the user should be able to specify a start and stop time, along with the number of frames to be generated within that time span. Given these inputs, the system would output the required number of images to produce the specified "movie".

i. Robert Wolff has observed that "...3 dimensional volumetric data ...involve visualization techniques that do not currently exist in a low-cost workstation environment. Nevertheless, without using these methods, it is difficult, if not impossible, for a scientist to gain a good understanding of a wide variety of data ...(26:35)". In order to permit usage by the largest possible number of engineers and researchers, a system should be designed for use at a workstation. It should also be as portable as possible to allow its use on a variety of systems.

1.4 Scope

Since this thesis is an initial attempt to use volumetric techniques to display flow field data, I will not be developing the entire graphics system outlined above. Instead, I will design and implement what I consider the basic, most essential features of the system.

The first capability I will provide is the volumetric display of the flow field data along with the object under test. I will also implement mechanisms to allow the user to specify the desired viewing geometry along with any specific subvolume of interest.

Another capability I will provide will allow the user to select target values, value variances, and color-value associations (see specifications c, d, and e above). I will also provide a color key in the image.

Due to time constraints, I will not attempt to provide the facilities for specifying parameter interrelationships or automatically generating temporal feature display sequences (system capabilities f and h, respectively). The first capability would require the design and implementation of a command language capable of allowing the user to both select parameters and describe how they are to be manipulated to derive a new parameter. Instead, I will limit the system to displaying one scalar parameter per image. The second capability (time frame specification) will be omitted for two reasons. The first (and most critical) reason is that I don't believe there is sufficient time available for me to develop this feature. Secondly, once implemented, the system I build will be able to generate a time sequence; however, the user will have to specify each frame individually. Although this is a more tedious method, it will produce the desired results.

Finally, I will strive to make the system as portable as possible. This may require foregoing system-specific utilities in favor of writing more generalized routines; however, the gain in supportability should offset this cost.

1.5 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 outlines the decisions required to design the system outlined in Section 1.1. In this chapter, I discuss the alternative methods I considered, then explain why I selected the options I used.

Chapter 3 describes the implementation of the imaging system. Along with a general overview of the system organization, specific key algorithm details are discussed.

Chapter 4 summarizes my thesis results along with recommendations for future system uses and enhancements.

Finally, a user's guide for my system, the Volumetric Imagery Program for Engineering Research (VIPER) is included as an appendix.

II. System Design

Dr Joseph Shang, an aeronautical engineer at the Air Force Wright Aeronautical Laboratory (AFWAL), recently noted that "to keep up with the rest of the world ... we need to develop a good three-dimensional graphics capability for displaying flow fields (21)". A panel report to the National Science Foundation reaches much the same conclusion, noting that "there are many known structures associated with fluid flows such as shock waves, vortices, shear layers and wakes ... Software algorithms and visualization techniques are needed to identify these structures in 3D flow field solutions ... Innovative techniques for displaying 3D data are required both in hardware and software (18:A13)". I believe that a relatively new volumetric rendering technique presents a solution to this visualization problem.

Marc Levoy recently outlined a technique he used to generate images of molecular structures (15:29-37) and noted that most previous systems approximated surfaces by arranging geometric primitives (such as triangles or rectangles) in such a way that they were "close" to the desired surface. As earlier noted, this technique has several major drawbacks. Volumetric rendering (VR), on the other hand, is well suited to the computational fluid dynamics (CFD) field.

2.1 Data Structure Selection

One data structure used in VR is very similar to the data generated by flow field analysis. This data structure is a simple 3D array ("spatial enumeration") consisting of data points, each of which contains discrete flow field parameter values. The alternative data structure for VR use is the octree (9:17).

Octrees are tree structures that have eight children per parent node. To build an octree, a recursive routine is used to dynamically subdivide volumes into octants until a predefined level of data homogeneity within each subvolume is achieved (11:15). Once this structure is built, it may be easily manipulated and traversed for image generation (20:100).

When short execution times are desired and 1,000,000 or more data points are to be processed, efficient data manipulation is a key concern. Upon reading Sandor's octree description and algorithms (20), the octree was my initial choice for the primary data structure in my system. However, Fujimoto, Tanaka, and Iwata have shown that the octree's efficiency grows considerably worse than that of spatial enumeration as the depth of the volume being rendered grows (9:21). Since my objective is to render an image of an entire flow field, I anticipated passing rays through a volume with a depth of 100 or more. Based on the findings of Fujimoto *et al*, I decided to use spatial enumeration as the principle data structure for my system.

2.2 Object Injection

Having selected spatial enumeration for the system, the next step was to select the best method for rendering the object under test (OUT) in the same image as the flow field data. Fortunately, the aerodynamic engineers' method of generating flow field data suggests a simple method for identifying the OUT within the flow field. In order to develop their data, the engineers build a 3D digitization of the OUT and use this model to build their volume of flow field data (2). Thus, by using spatial enumeration as the basic data structure for my system, the method of OUT representation is also established. If the engineer who develops the digitized model of the OUT also uses a unique data value to identify the presence of the OUT at a data node, he may also select that value as a target value to be sought and rendered by the Volumetric Imagery Program for Engineering Research (VIPER). This capability will also provide the option of rendering an image of the flow field with or without the OUT being displayed. It should be noted, however, that this method represents a double-edged sword. While it provides a simple and versatile solution to the problem of displaying the OUT within the flow field, the data value used to represent the OUT must be unique within the flow field; otherwise, portions of the flow field will be rendered with the same color used to denote the OUT and an element of ambiguity may be introduced into the image.

2.3 *Portability*

Since VIPER is intended for eventual use by the engineering community in general, system portability was another critical consideration. To be portable, the system needed to be implemented with a general purpose high order language and as free as possible of system-dependent features. Once installed, the system also had to be as simple as possible to maintain. Since many graphics tools are written in the 'C' programming language, I decided to implement my system in 'C'. Portability was also enhanced by using the Utah.rle graphics routines. These consist of a public domain image file format and a set of tools to reposition and combine images. Use of this system provided a common, inexpensive, and pre-established base for space-efficient image description.

In order to be portable, user interfaces also had to be general in nature. Although mouse- or icon-based interfaces would be simpler for the user, their use would severely limit system portability. Instead, I chose simple question/answer prompts for user interfaces. This approach requires only a display device and a keyboard.

Finally, all system-specific parameter values (such as the pixel dimensions of the display device being used) would be isolated in a separate header file. This would provide a central file that could be modified to tailor VIPER to any particular system being used.

Having made the major design decisions, I began implementing a system that would satisfy the goals set forth in Chapter 1. The next chapter details how this program is implemented.

III. System Implementation

The Volumetric Imagery Program for Engineering Research (VIPER) provides the capability for rendering full 3D images of aerodynamic flow field data. Chapter 2 outlines the preliminary design decisions that were required before program implementation began. This chapter details how the actual program is constructed and how it performs its tasks. After a brief overview of the viewing geometry involved in volumetric rendering, each of VIPER's subfunctions will be examined. The subfunctions are discussed in the same order in which they occur during program execution.

3.1 Viewing Geometry

Before discussing the implementation details of VIPER, it is necessary to understand the viewing geometry involved. The viewing geometry is determined by the locations of the viewer's eye, the viewing plane, the data volume, and all light sources selected by the user (see Figure 3).

3.1.1 Viewer's Eye. The viewer's eye is represented by a single point in space. Its location is selected by the user. All of the rays that are cast to form the final image emanate from the viewer's eye. The direction of the emanating rays is determined by the location of the viewing plane.

3.1.2 Viewing Plane. The viewing plane is an array of picture elements (pixels). The size and location of the viewing plane are user-selectable. A ray is cast from the eye through the center of each pixel. Once the ray is defined, it is extended as far as necessary to determine whether or not it intercepts the data volume.

3.1.3 Data Volume. The data volume may be thought of as a box of data suspended in space. This box is comprised of volume elements (voxels) arranged in a rectangular, 3D mesh. The location of the data volume is user-specified by entering the desired coordinates for the centerpoint of the volume. Fixing this point also defines the planes that form the faces of the volume. In this respect, spatial enumeration acts as its own bounding structure, a graphics technique used to improve algorithm efficiency.

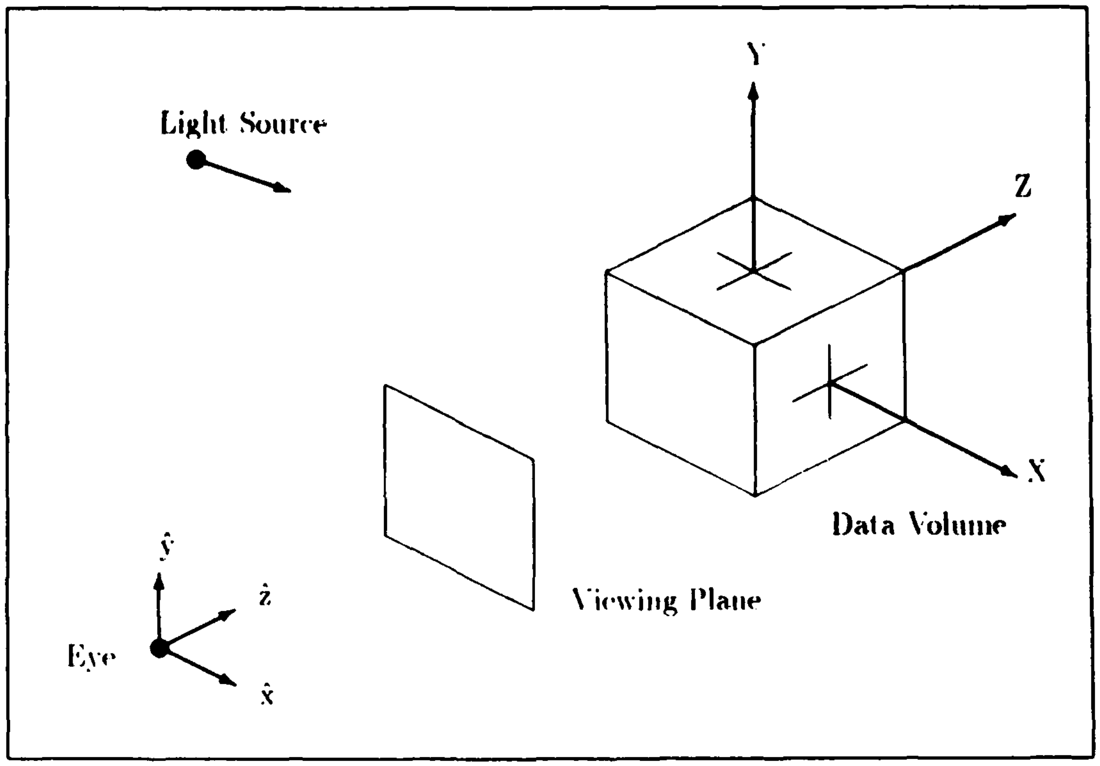

Figure 3. Initial Viewing Geometry

3.1.4 Light Sources. The final element of the viewing geometry, light source location, is another user-specified feature. For each light source, the user must provide the desired location coordinates. Each light source is modeled as a point that emits parallel rays of white light. This model was selected for its simplicity. Other models may provide a more realistic approximation of light's behavior; however, they also require a higher computational overhead.

3.1.5 Initial Setup. The initial viewing geometry may be looked at from two different perspectives: the user's and the programmer's. Although they are very similar, they have one distinct difference. The user's perspective is centered about the viewer's eye; however, as shown in Figure 3, the program actually revolves about the center of the data volume.

To the user, the initial viewing geometry places the origin at the viewer's eye and all manipulations are based on the $\hat{x}$, $\hat{y}$, $\hat{z}$ axes as shown in Figure 3. The viewing plane is centered on and orthogonal to the $\hat{z}$ axis at a user-specified distance from the viewer's eye. Likewise, the data volume is initially centered orthogonally on the $\hat{z}$ axis at a user-specified distance from the eye. Finally, the light sources are suspended in space at their user-specified coordinates.

On the other hand, the programmer sees the initial geometry with the data volume's center point at the X, Y, Z axes origins (see Figure 3). Although the viewer's eye is still pointed directly down the Z axis, it is moved in the *negative* Z direction to establish the user-specified distance from the viewer's eye to the center of the data volume. Once the viewer's eye position is defined, the viewing plane is centered about the Z axis at the proper distance from the eye. Likewise, the Z-ordinate of each light source's position is adjusted relative to the eye's Z-ordinate.

3.1.6 Viewing Transformations. In order to provide the capability of viewing all portions of the flow field, the standard graphics viewing transforms (rotation, translation, and scaling) must be available. The difference between the user's and the programmer's perceptions of the viewing geometry arose from the implementation of these transforms. From the user's point of view, the *data volume* is being rotated, translated, and scaled. However, transforming the data volume would require changing each voxel's location. A typical volume may contain more than a million voxels; at 18 mathematical operations (nine multiplications and nine additions) per voxel this approach would require a great deal of computation time.

Therefore, the rotation and translation operations are performed on the viewer's eye position, each light source's location, and each pixel coordinate. In order to achieve the results desired by the user, the objects are transformed in the opposite sense to the user's entries. For example, if the user wishes to rotate the data volume 45° about the X axis, VIPER will rotate the viewer's eye, all light sources, and all pixels −45° about the X axis. Thus, the volume remains fixed while all the other components in the viewing geometry are moved; however, the illusion of changing the volume's orientation is maintained for the user's convenience.

Unfortunately, the scaling transform could not be effected by operating on the eye and pixel positions; the volume itself needed to be transformed. However, the nature of the volume allowed me to scale it without modifying each voxel. The key to my solution lies in the fact that, although spaced irregularly, the voxels' locations are otherwise uniform; that is, they form distinct planes. Therefore, rather than storing a distinct set of coordinates for each voxel, I was able to save all X ordinates in a 1-dimension array; all Y and Z locations were stored likewise. This approach allows me to represent the locations of many voxels by using relatively few numbers (*e.g.*, a million voxels may be defined with as few as 300 numbers). Each location ordinate must also be adjusted to fit in with the viewing geometry; this step involves subtracting half the volume's span from each ordinate value (*e.g.* if the volume's X ordinates range from 2 to 6, 4 would be subtracted from each ordinate. The resulting span would go from -2 to 2 and be centered about $X = 0$.) When the user wishes to change the size of the volume, all that is required is a simple multiplication of each ordinate by the user-specified scaling factor.

3.2 Target Parameters

In addition to specifying the viewing geometry, VIPER allows the user to select target values of interest, the red, green, and blue (RGB) color values to be associated with each target value, the maximum allowable deviation from the target values, and the maximum opacity for surfaces that correspond with the target values. These parameters are stored as a record; a separate record is maintained for each target value selected.

3.2.1 Target Values. VIPER allows selection of up to five target data values. If the user wishes to have the OUT rendered in the image, one of the target values must be the data value that was used to designate the OUT in the volume.

3.2.2 Target Colors. A separate RGB color triplet is specified for each target value selected by the user. When specifying RGB triplets, the user must take care to select a different triplet for each target value. Failure to do this will result in surfaces of different isovalues being rendered with the same color; image ambiguity will result. Furthermore, the RGB triplets should be distinct enough that color variations resulting from surface highlighting will not create a color crossover (where surfaces with different initial colors appear to have the same color). RGB triplets are expressed as sets of three real numbers that may range from 0.0 to 1.0.

3.2.3 Allowable Value Deviation. The allowable value deviation defines the "window" of values that will be rendered. In order to provide visual clues about the proximity of a surface's value to the target value, surface samples with values that are near the center of the value window will be more opaque than samples that fall near the window edges.

3.2.4 Opacity. The final user-controlled parameter is the maximum surface opacity. This parameter may range from 0.0 to 1.0, with higher values denoting a higher degree of opacity. An opacity of 0.0 will produce surfaces that are entirely transparent and colorless while a value of 1.0 will be completely opaque. As mentioned previously, the final opacity of a surface sample will depend upon its value's proximity to a target value. This opacity is determined by the following equation:

$$opacity_{sample} = opacity_{max} \times \left(1.0 - \frac{|\, value_{target} - value_{sample} \,|}{allowable\ deviation}\right) \qquad (1)$$

for any $value_{sample}$ that falls within the specified window of values. All samples that do not fall within the value window will be assigned an opacity of 0.0.

It should be noted that this opacity formula differs from that of Levoy's (15:32). The reason for this difference is the fact that Levoy is trying to render surfaces of a given thickness with the following formula:

$$opacity_{sample} = 1 - \frac{1}{thickness} \times \frac{|\, value_{target} - value_{sample} \,|}{|\, local\ gradient \,|} \qquad (2)$$

After experimenting with both equations, I found that images rendered using Equation 2 showed the same structures as images rendered with Equation 1. However, Equation 2 produces a dimmer image. Due to the brighter images, along with the fact that it is computationally faster, VIPER uses Equation 1 to determine surface opacities.

3.3 Ray Casting

As previously noted, each viewing ray emanates from the viewer's eye and is directed through the centerpoint of a pixel in the viewing plane. These two points determine the direction of the ray. Once the ray is formed, it is normalized to a magnitude of unit length. The ray originates at the pixel through which it passes.

The first step in ray casting (after the ray is formed) involves determining if the ray intercepts the data volume. A two-step process is required to perform this check. First, the ray's point of origin is tested to see if it is located *within* the data volume. If this is the case, the ray is assumed to intercept the volume with an impact point that coincides with the ray's point of origin. If the ray doesn't originate within the volume, the next step is to see if the ray extends into the volume. To perform this check, the distance from each face of the data volume to the ray's point of origin is divided by a directional component of the ray (for instance, the distances from the ray's origin to the left and right faces of the data volume are divided by the X-component of the ray). Once all six face-to-eye distances are calculated, each ray component is multiplied by one of the distances and added to the corresponding eye positional ordinate to yield a planar impact coordinate. If this point is on one of the volume's faces, the distance is saved. When all six distances have been checked, the nearest ray-volume impact point is selected as a starting point for volume traversal. If no intersection points are found, the ray misses the data volume and processing proceeds to the next pixel.

If a volume impact point is found, the ray is incrementally extended through the data volume with the data being tested at each step. Levoy's approach to this process involves "resampling the ray at K evenly spaced locations ... along the ray (15:30)". An early version of VIPER used this approach with disappointing results (see Figure 4). This technique can exhibit strong aliasing artifacts and may omit entire voxels. The reason for this lies in the fact that the probability of finding all surfaces of interest depends on the length chosen for the intervals between steps. If a large step is taken, many interstitial surfaces may be missed. If a smaller step is taken to account for this problem, the algorithm will experience a linear increase in execution time (e.g., doubling the sampling frequency will also double the time required to traverse the volume.). Finally, it is impossible to assure that all surfaces of interest will be found; to do so would require taking an infinite number of samples along each ray. As an alternative, I have developed the following algorithm.

To start the volume traversal process, the ray-volume impact point is also used as the first ray-voxel intersection point. The point where the ray emerges from the voxel is then calculated. Once these two points are found, the intervening data may be tested against the user-specified target values (as described below). Once the current voxel has been processed, the ray moves to the next voxel by

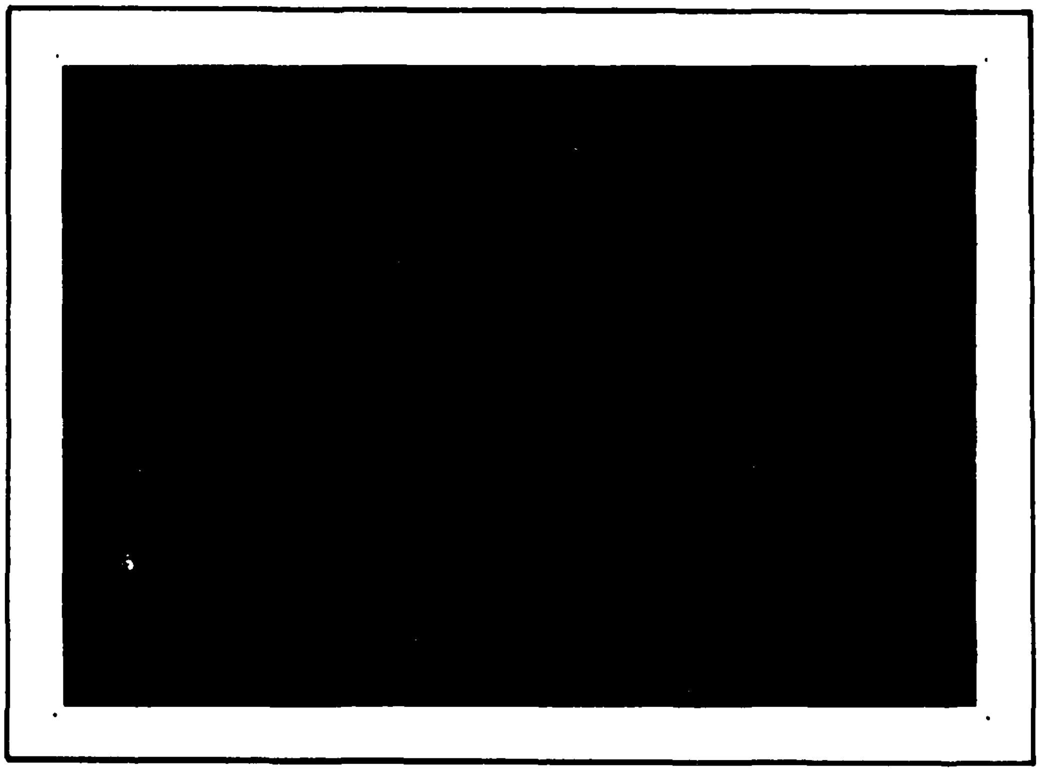

Figure 4. Aliasing Produced by Sampling at Even Intervals.

assuming that the current exit point coincides with the entry point into the next voxel (see Figure 5). This process continues until either the ray emerges from the data volume or the light available to the ray is depleted.

3.4 Data Sampling and Color Calculations

Once a ray is positioned within a voxel, the data lying along the ray's path must be tested against the target values selected by the user, classified, and (if appropriate) factored into the ray's color. This sequence forms a pipeline as described by Marc Levoy in (15).

3.4.1 Trilinear Interpolation of Data Values. The first step in testing the data is determining the data values at the ray-voxel entry and exit points. This is accomplished by following Levoy's suggestion (15:30) and trilinearly interpolating

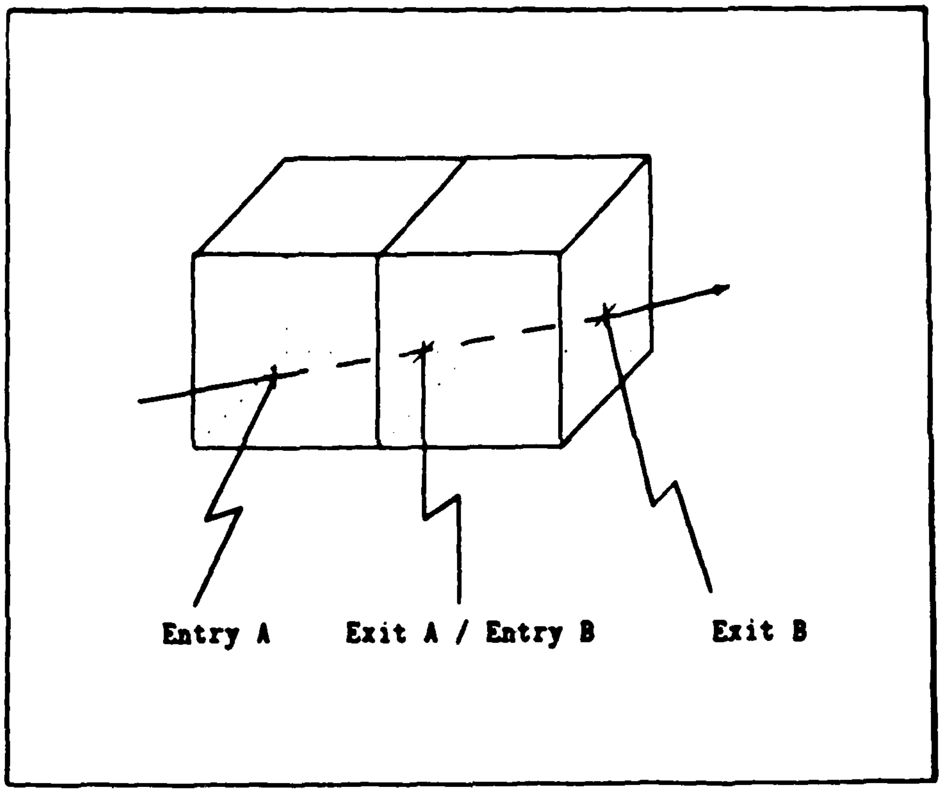

Figure 5. Ray Traversal Through Adjacent Voxels

the values at these points based on the data values at the voxel's vertices. Once these two values have been determined, each target value can be tested for inclusion along the ray segment defined by the entry and exit points.

There are three possible categories that a target value may fall into:

- It lies between the values of the two endpoints.
- It doesn't lie between the values of the two endpoints but one of the endpoints lies within its window of allowable values.
- It doesn't lie between the values of the two endpoints and neither endpoint lies within its window of allowable values.

If a target value falls into the latter category, it may be disregarded for the current voxel. However, if it falls into one of the other two categories, the target value must be parameterized into a relative distance term as follows:

If the target value falls between the two endpoints,

$$t = \frac{value_{target} - value_{entry\,pt}}{value_{exit\,pt} - value_{entry\,pt}}. \tag{3}$$

However, if the target value falls into the second category above, t will equal 0 or 1 depending on which endpoint falls within the window.

After all target values have been classified and parameterized, they are sorted by ascending values of t. After the sorting is complete, each target value's effect upon the ray's color may be calculated.

Since this technique involves locating the surface by relating its value to the values of the ray segment's endpoints, the algorithm is freed from the constraints of sampling points along the segment. As a result, I have eliminated the aliasing problem I encountered using "K evenly spaced locations". Also, since the ray is being moved along a voxel at a time, the maximum number of times the ray must be advanced and calculations made is directly related to the number of voxels in the volume instead of the dimensions of the volume itself. Thus, for a 100 x 50 x 30 volume, a ray will need a maximum of only 100 extensions; it makes no difference whether the volume has a span of 1,000 or 1,000,000 units. On the other hand, if VIPER used evenly spaced samples, increasing the volume's span by a factor of 1,000 would also increase the number of samples required by a factor of 1,000.

3.4.2 Sample Color and Opacity Assignment. Determining the RGB values to be used for each target value is simple: the user has already defined the RGB values associated with each target value. Determining the opacity of the surface is little more involved. If the t-value associated with the target value is greater than 0, the opacity of the surface is equal to the user-specified opacity associated with the target value. If $t = 0$ the surface opacity is found by substituting $value_{entrypoint}$ for $value_{sample}$ in Equation 1. If $t = 1$ the target value is disregarded; it will be accounted for when the next voxel along the ray's path is processed.

3.4.3 Surface Highlighting. The next step in the rendering pipeline involves determining the surface highlights due to the light sources selected by the user. I use the Phong shading model to determine surface highlights. However, before the Phong model can be used, the normal to the surface at the point of ray intersection must be determined. I use the local surface gradient to determine the surface normal. According to Levoy (15:31), the local surface gradient can be determined by

$$\begin{aligned}\nabla f(x_i, y_j, z_k) \approx \; & (\tfrac{1}{2}(f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)), \\ & \tfrac{1}{2}(f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)) \\ & \tfrac{1}{2}(f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})) \qquad (4)\end{aligned}$$

This approach assigns the same surface gradient to all points within a given voxel. VIPER extends this technique and uses bilinear interpolation to approximate local gradients (and thus the surface normal) as follows:

$$\begin{aligned}\nabla f(surface_{x,y,z}) \approx \; & ((f(x_{i+1}, y_{surface}, z_{surface}) - f(x_i, y_{surface}, z_{surface})), \\ & (f(x_{surface}, y_{j+1}, z_{surface}) - f(x_{surface}, y_j, z_{surface})), \\ & (f(x_{surface}, y_{surface}, z_{k+1}) - f(x_{surface}, y_{surface}, z_k))) \qquad (5)\end{aligned}$$

Although VIPER's technique is more complicated than Levoy's, it gives a more accurate value for the surface normal at a given point.

Once the surface normal is calculated, the Phong shading model may be applied to the surface to determine surface highlights.

3.4.4 Light Attenuation. Light attenuation presented a special problem: although some method was required to model the effects of light passing through surfaces of different opacities, an accurate and realistic model would be extremely complex and slow. Also, as Michael Keeler notes, "it's much more important to represent information than to go for *realism* in our field (25:25)." Therefore, as an alternative to an intricate model of light-surface interaction, I developed the following method to account for decreasing light levels. Unattenuated light is assigned a light level value of 1.0. As each pixel ray is formed, it has an available light level of 1.0. However, as the ray passes through surfaces, the light level is decreased with the following formula:

$$light\ level = light\ level \times (1.0 - opacity_{Surface})$$

This technique is based on the premise that light may only be reflected or transmitted; none may be absorbed by a surface. Another simplification I made assumes that surfaces only reflect from the side that faces the viewer. Thus, if light is reflected from an underlying surface, all of the reflected light will pass through any intervening surfaces. A final assumption I made is that all surfaces are perfectly smooth; no light is scattered through reflection. Although these assumptions grossly simplify the interaction between light and physical objects, the resultant model does produce the effect of light attenuation without slow, complex computations.

3.4.5 Combining Sample and Ray. The final step to be performed involves factoring the color of the surface into the final color of the ray. To do this, VIPER simply multiplies the RGB values of the surface by the opacity of the surface and the available light. This color is then added to the ray's color. Thus,

$$RGB_{Ray} = RGB_{Ray} + Light\ Level \times RGB_{Surface} \times Opacity_{Surface}$$

and the final color of the pixel through which the ray passes is given by

$$RGB_{Pixel} = \sum_{i=0}^{n}(Light\ Level_i \times RGB_{Surface_i} \times Opacity_{Surface_i}).$$

By decreasing the light level and incorporating each surface's color contribution to the pixel as the ray is traced through the volume, I am able to perform a front-to-back summation and thus avoid the extra time and overhead required by the front-to-back traversal followed by a back-to-front summation technique developed by Frieder, Gordon, and Reynolds (8).

3.5 *Summary of Algorithm*

- Read in Data and Build the Data Volume.
- Interactively read in User-Controlled Parameters.
- Transform the Viewing Geometry.
- For Each Pixel:
  - Form Ray.
  - Check for Ray-Volume Intersections. If Ray Intersects the Volume, Trace the Ray Through the Volume. At each Voxel:
    - Locate Ray's Entry and Exit Points. Trilinearly Interpolate their Values.
    - Locate all Ray-Target Value Surface Intersections.
    - Calculate Each Surface's Normal, Shading Factor, Contribution to Pixel's Final RGB Value, and Light Attenuation.
    - Extend Ray to Next Voxel.
  - Save Pixel's Final RGB Values in a Frame Buffer.
- Save Frame Buffer in Utah.rle Format.

IV. Conclusions

The first three chapters of this thesis present the rationale and implementation details involved in VIPER's development. In this chapter, I will assess VIPER's capabilities and performance characteristics. I will also outline what I believe would be major enhancements to VIPER as well as other fields in which VIPER might prove useful.

4.1 Performance Characteristics

4.1.1 Expert Opinion. Inasmuch as I am not an aerodynamicist and therefore not qualified to assess VIPER's utility in that field, I sought the opinions of those who would know what is useful and what is not. Accordingly, I presented VIPER-generated images of theoretical flow field data to Capt Beran (an instructor in the AFIT Department of Aeronautical Engineering) and Dr. Webster, an aerodynamicist from the Air Force Wright Aeronautical Laboratory. Both of these reviewers made favorable comments on VIPER's imagery and capabilities.

In an effort to demonstrate VIPER's applicability to other fields, I used VIPER to produce a series of images based on the atomic energy levels surrounding compressed crystalline structures. Lt Col Lupo, a professor from AFIT's Physics Department, was favorably impressed by these images, noting that they closely corresponded with the 2D contour plots he had developed for the data. He furthermore stated that the 3D images developed by VIPER were "much better" than the polygonal-based images he had been using.

Finally, VIPER was used by Capt Gary Lorimor (a fellow AFIT student) in the course of his thesis efforts. When he noted errors generated by a Polhemus Navigation Sciences 3-SPACE tracker, he took a series of readings at specific spatial intervals. Using these readings, VIPER was able to generate an image of predicted error values within the space where the tracker was being used. Studying the images allowed Capt Lorimor to identify specific regions of both high and low error zones.

Based on these assessments, I believe that the applications of volumetric rendering should be further explored, not only in the field of computational fluid dynamics but in other fields of study as well.

4.1.2 Algorithm Efficiency. VIPER is a graphics program that uses a ray-casting algorithm to provide full, 3D images of a volume of data. In order to assess algorithm efficiency, one must normally specify whether efficiency is being expressed in terms of time or space complexity. In assessing VIPER, I have found that this distinction is not necessary; the algorithm's time *and* space complexities are both dependent upon the size of the data volume (in terms of voxel dimensions) to be rendered.

If one disregards the process of reading in the data volume, VIPER operates with an efficiency of $O(pr)$, where p is the number of pixels through which rays are being cast and r is the volume's largest dimension (in terms of voxels per edge). The efficiency notation for the data-reading portion of VIPER depends on the unit of measurement to be used. If measured in terms of total number of mesh nodes in the data volume, this segment operates at a linear efficiency level of $O(d)$ (where d is the number of data points to be read in). On the other hand, if one uses the volume's dimensions (measured in voxels per edge) as the standard of measure, VIPER's data-reading performance may be expressed as $O(ijk)$ where i, j, k are the X, Y, Z dimensions of the data volume.

4.1.2.1 Execution Time. It is difficult to give a meaningful figure for VIPER's execution time due to the number of factors (*e.g.*, number of voxels in the volume, pixel dimensions of final image, object magnification, number of surfaces being sought, surface opacity, etc.) involved. However, to produce a demonstration video that is thirty seconds in length required the rendering of 900 images. The volume rendered had voxel dimensions of 100 × 68 × 68 for a total of 162,400 mesh nodes; the video images showed the volume at different angles and different opacity settings for the target surfaces. Each image measured 512 × 512 pixels. Using the Digital Equipment Corporation GPX Microvax III graphics systems located at AFIT, the time required per image ranged from 27 minutes to approximately 3.5 hours. Although these figures obviously preclude the use of VIPER for interactive image generation and viewing, they are not prohibitively large; 2D images used to demonstrate graphics capabilities at NASA's Langley Research Center require "*only* two hours of central processing time (17:5)" on a VPS-32 supercomputer. Therefore, I believe that VIPER demonstrates a useful image production rate.

4.1.2.2 Memory Requirements. Memory requirements, like execution times, are difficult to quantify; therefore, I shall again resort to the memory requirements I observed during the production of VIPER's demonstration video. The storage requirement for the data file was 10.711 megabytes (Mb) of online memory; however, it should be noted that the data file was stored in ASCII format. While less memory would be required if data was stored in a binary format this would induce an element of system dependency and thus limit the portability of the program. While executing, VIPER required 2.317Mb; this requirement will change in proportion to the size of the data volume.

4.1.3 Data Display. As noted in Chapter 1, most aerodynamic images to date have been restricted to either 2D displays, 2D images superimposed upon each other in a 3D arrangement, color-coded regions on the surface of the object-under-test (OUT), or a 3D display of vectors or particle paths. Figure 6 shows a typical 2D rendering of the flow field around a frictionless sphere. In contrast, Figure 7 shows a stereo image of the same flow field as rendered by VIPER. The different shells displayed in the image correspond to various values of the velocity potential about the sphere. These shells are only hinted at by the long streamlines shown in Figure 6. However, careful comparison between these two figures shows that VIPER's image is exactly what is predicted in (13). Also note the colored numbers in the lower left corner of the image; they form the image color key and indicate the target values being rendered as well as the color with which they are being displayed. The pale grey sphere in the center of the display represents the OUT.

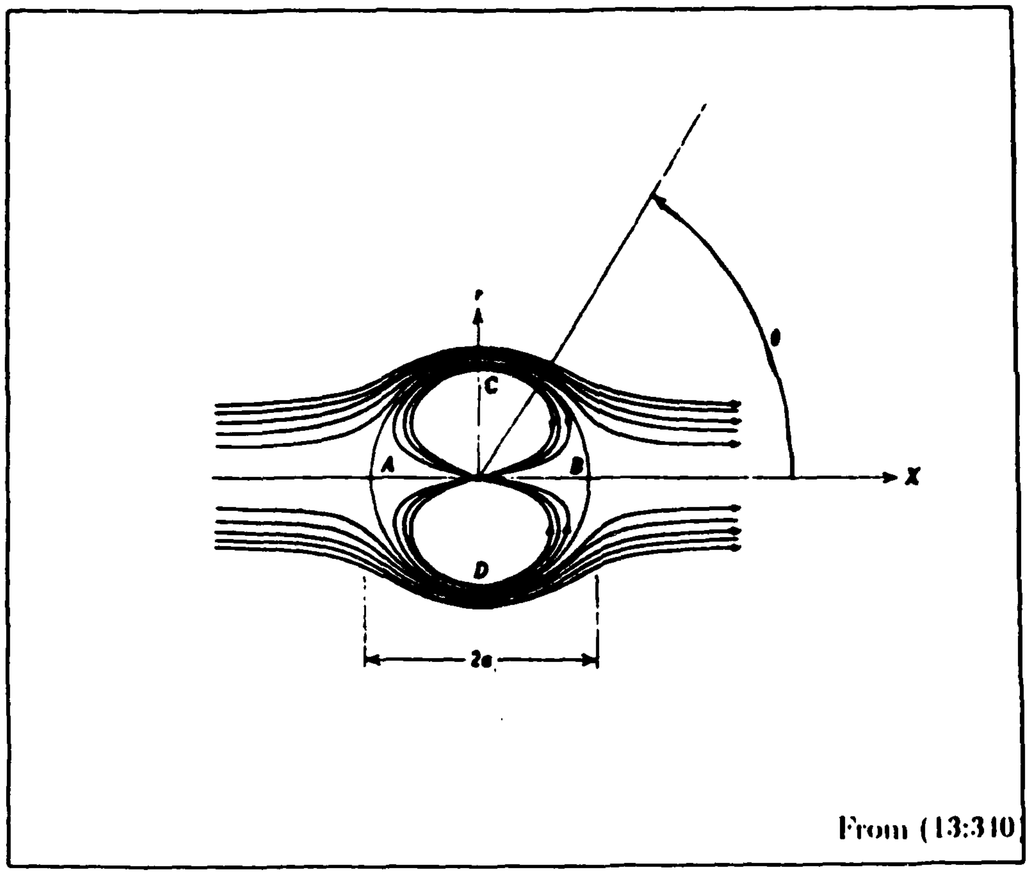

Figure 6. Textbook Image of Flow About a Sphere

VIPER's accuracy may also be assessed by comparing the contour plot of a wake vortex (Figure 8) with the VIPER-generated image of the same data set (Figure 9. The range of VIPER's capabilities is suggested by Figure 10 which shows the same view of the vortex rendered with different opacity values.

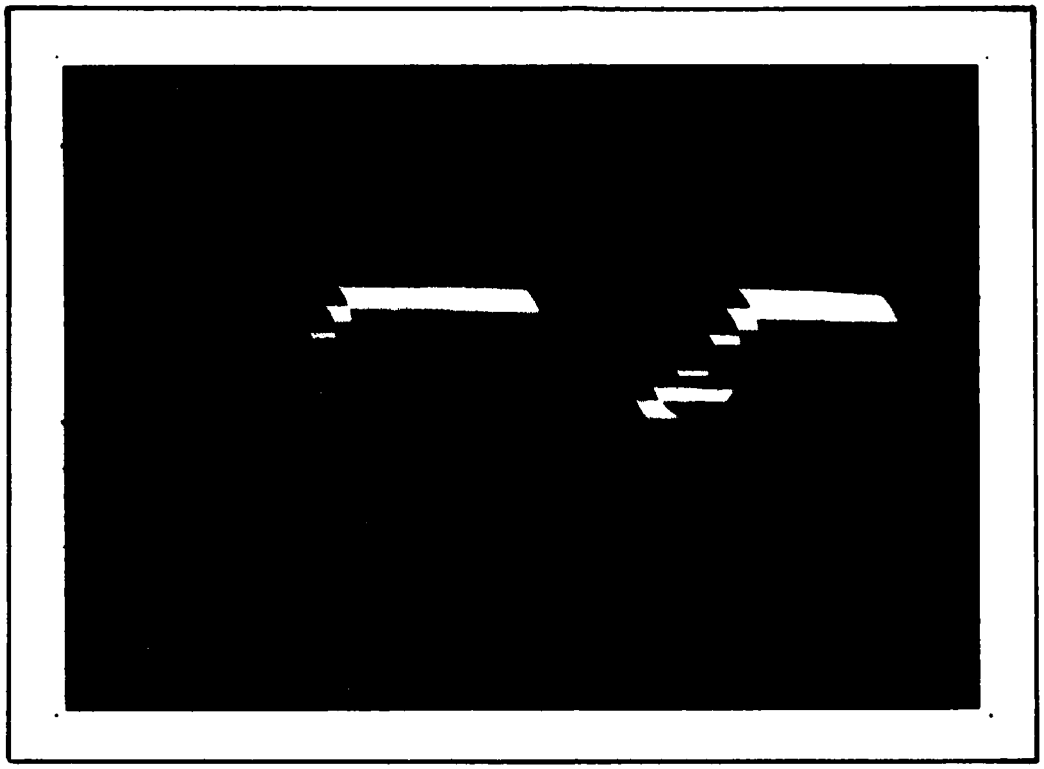

Figure 7. VIPER Image of Flow About a Sphere

I believe that these examples and the previously-cited reaction from potential users demonstrate VIPER's potential for improving the state of aerodynamic flow field imagery.

4.1.4 Aliasing Effects. As I noted in Chapter 3, the original strategy of sampling at even steps along each ray produced very pronounced aliasing artifacts due to the resultant gaps in coverage that occur between sampling points. Locating surfaces as a function of segment endpoint values eliminated this problem. However, there are still sources of aliasing that should be addressed.

To begin with, the number of voxels in the data volume itself will have an effect on the degree of spatial aliasing apparent in the final image. Specifically, a given volume will exhibit fewer aliasing artifacts if it is represented by 100,000 voxels than it would if it is comprised of only 100 voxels; this is another form of

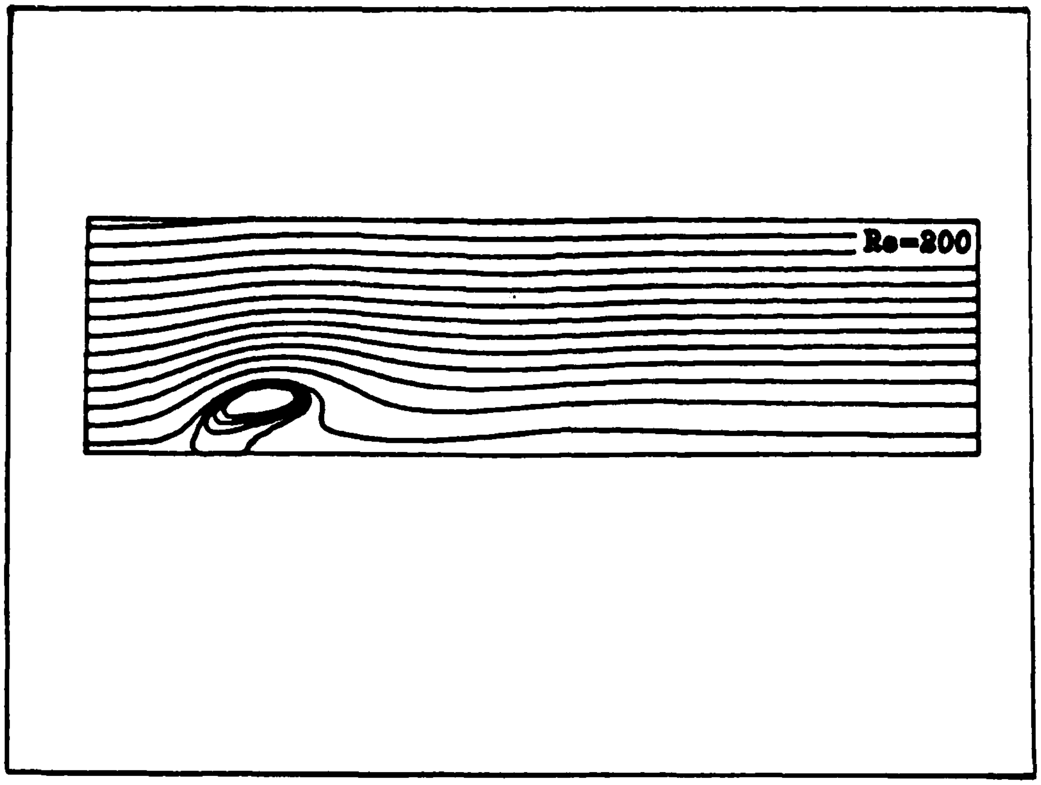

Figure 8. Contour Plot of Wake Vortex.

aliasing as a function of sampling frequency. However, the number of voxels in the volume will be dictated by the user's needs and system capabilities; this is not a problem that can be solved within VIPER.

Another aliasing effect that varies with the size of the voxels being rendered is also influenced by the steepness of the local gradient surrounding a given voxel, the size of the value window being used, and the opacity of the target surface. Given the right circumstances, the entire voxel will be colored and displayed as a rectangular block within the final image. This effect is most likely to occur when using a small value window and high opacity coefficient to render a region comprised of relatively large voxels containing a steep local gradient. The user may dampen this form of aliasing by specifying fairly wide value windows and low opacity coefficients, particularly when setting these values for rendering the OUT.

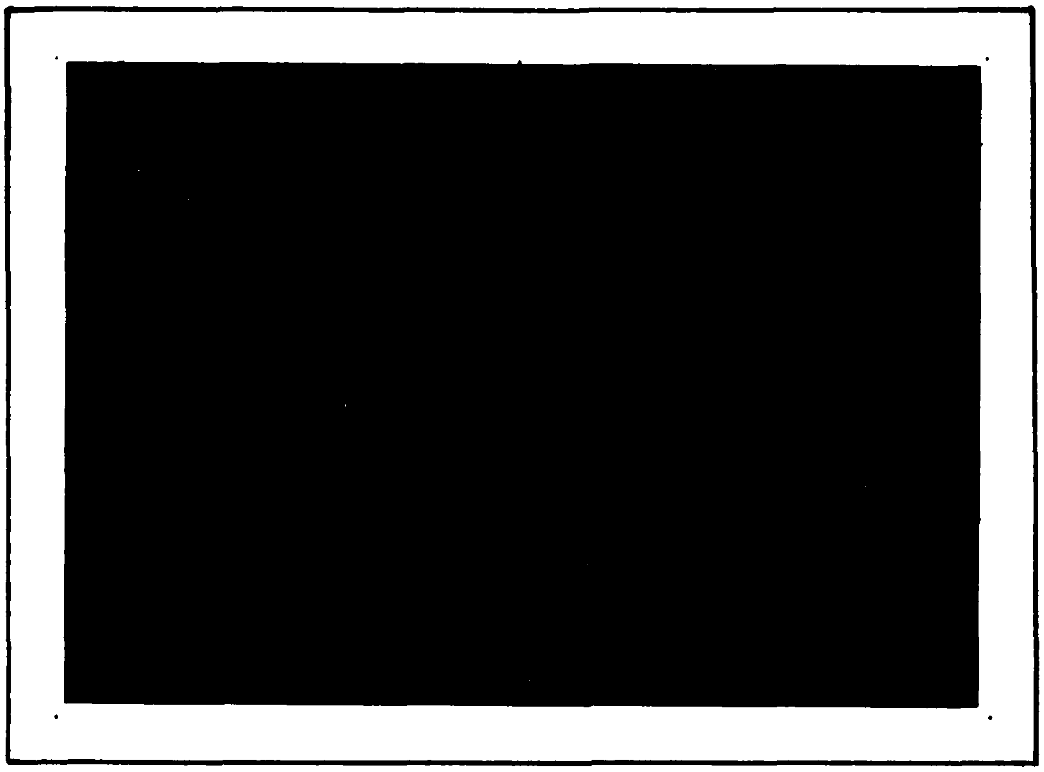

Figure 9. VIPER-Generated Image of Wake Vortex

Another source of aliasing is produced by point sampling and is dependent upon the resolution of the viewing plane (measured in pixel dimensions). The coarser the viewing plane, the more pronounced the aliasing (which manifests itself in the form of jagged edges) will be. This is due to the fact that the color of an entire area (the pixel) is determined by the color of an infinitely small area (the cross section of the pixel ray). Many schemes have been advanced in an effort to remedy this common graphics problem. Typical solutions include supersampling (firing more than one ray per pixel and averaging the resultant RGB triplets), filtering, and "blurring" the image by averaging the colors of adjacent pixels. All of these approaches incur a higher computational overhead and tend to degrade the sharpness of the final image. I experimented with supersampling and found that, although a linear increase in processing time was required, little improvement was achieved by firing four rays per pixel.

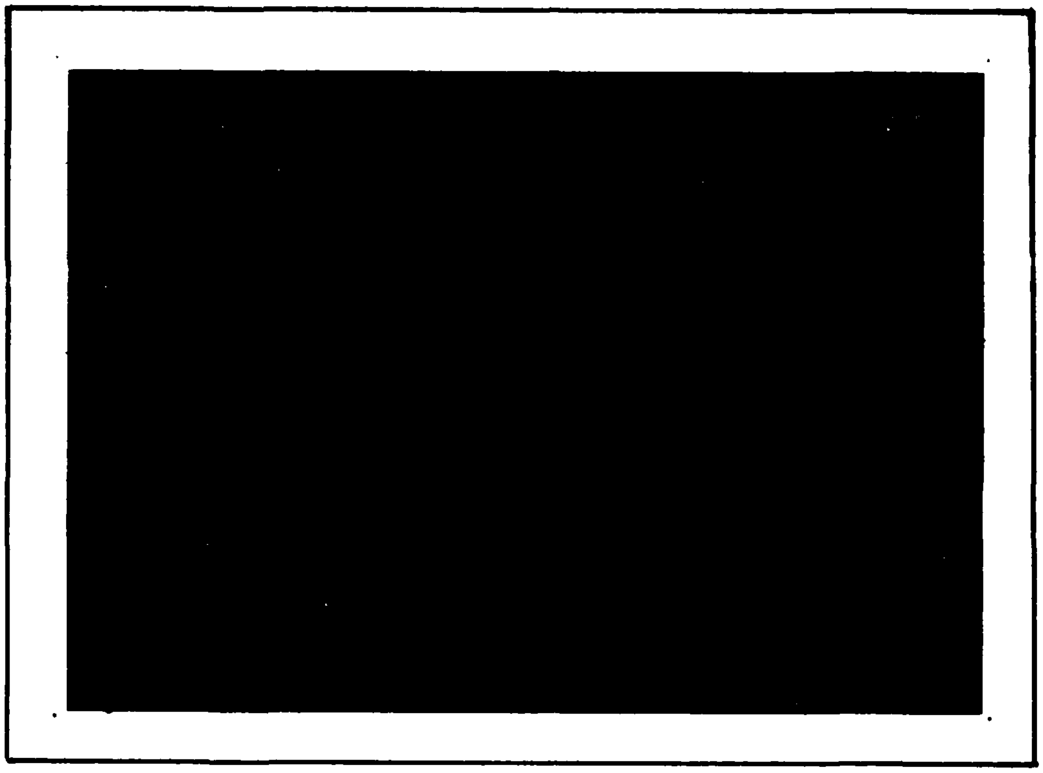

Figure 10. Wake Vortex Rendered With Different Opacity Values.

A final source of aliasing is identified by Drebin, Carpenter, and Hanrahan who note that "when rays diverge they may not sample adjacent pixels (5:72)" (Since a ray is fired through *every* pixel, I believe this is an error and *pixel* should read *voxel*.). However, giving the user the ability to adjust the the locations of the viewing plane and data volume provides the means to minimize this problem. As the distance between the viewer's eye and the viewing plane increases, the angle of divergence between rays decreases. As the distance approaches infinity, the ray divergence approaches zero. As an example of this technique, when making my demonstration video, I used a 512 × 512 pixel image plane set at a distance of 5000 units from the viewer's eye. Given this configuration, the maximum divergence between neighboring rays is only 0.0162°. If less divergence is desired, the image plane and data volume may be moved to an even greater distance.

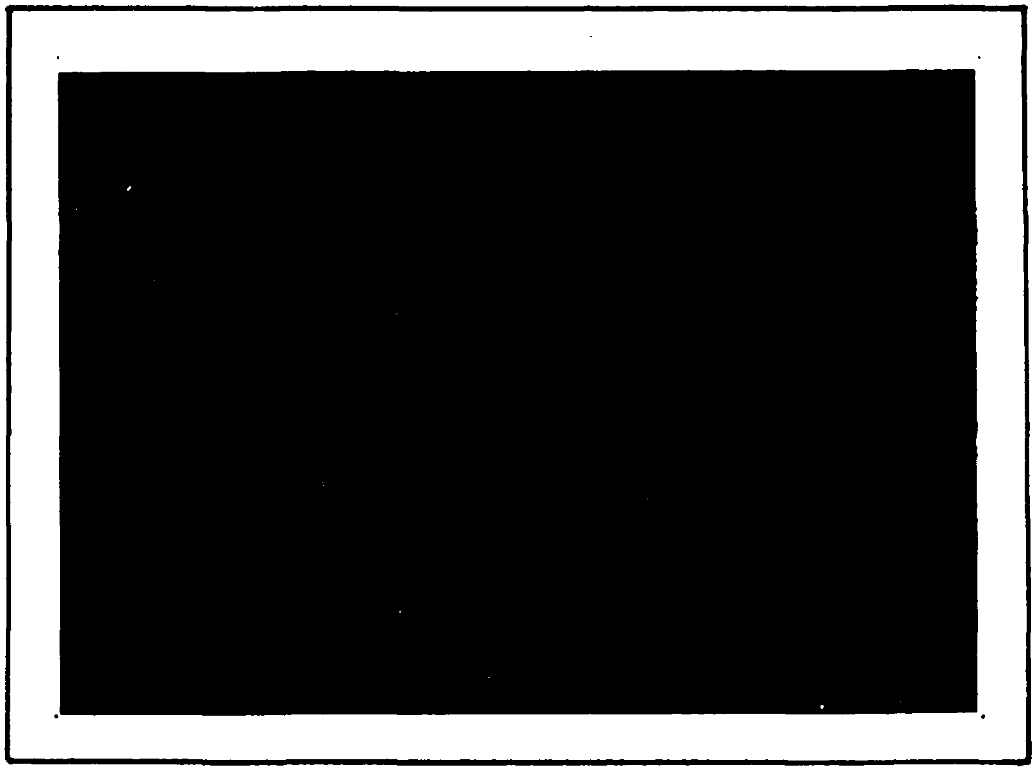

Figure 11. Subvolume Isolation Via Cutting Plane Method.

4.1.5 Parameter Control. There are a number of parameters that are explicitly controlled by the user. These fall into one of two general categories: scene specification and target value/property definition. After reading in the data volume, VIPER prompts the user to enter values for each control parameter.

4.1.5.1 Scene Specification. This first category of parameters allows the user to "set the scene". The user must first specify the viewing geometry. The parameters that control the initial geometry are the distance from the viewer's eye to the center of the data volume, the distance from the viewer's eye to the viewing plane, and the location of all desired light sources.

The user may also selectively "cut away" portions of the data volume in two ways. This permits the user to isolate a specific subvolume of interest. The first method of volume elimination is accomplished by placing the viewing plane within the volume, as shown in Figure 11. By firing pixel rays from the viewing plane instead of the viewer's eye, I was able to transform the viewing plane into a cutting plane. The second method of eliminating portions of the volume involves selectively removing user-specified slabs from any (or all) of the volume's six faces. Figure 12 shows the same flow field as Figure 7; however, 95% of the flow field has been selectively removed by taking slabs off of the front and back of the data volume. The central 5% of the volume is thus revealed for examination.

Once the subvolume has been defined, the user must input scaling factors, angles of rotation, and translational distances. Finally, the user is given the opportunity to change the size of the viewing plane. Once entered, these parameters define the final viewing geometry. Another variable to be entered specifies the ambient light level within the scene. Finally, an RGB color triplet must be specified for the image matte. This is a color to be applied to any pixel ray that misses the data volume. Rays that do intersect the volume are initially black (RGB = 0,0,0) and change as they encounter target surfaces. The objective of this feature is to display the silhouette of the data volume against a colored mat; this allows the user to orient the rendered surfaces within the volume itself. If the user wishes to disable this feature, he only needs to select a mat RGB triplet of (0,0,0).

4.1.5.2 Target Values. The current version of VIPER allows the user to specify up to five target values to be sought out and rendered. For each target value, the user must specify the value itself, a window of acceptable values which is centered about the target value, the RGB color triplet to be associated with the target value, and the opacity of the target value's surface. These values are assigned independently for each target value to be sought. However, the reflectivity values (diffuse and specular reflection coefficients) are specified once and applied uniformly to all rendered surfaces.

4.1.6 Portability. Since VIPER is intended for users who will have a wide variety of computing facilities at their disposal, I placed a great deal of emphasis on maintaining portability. Accordingly, I eschewed system-specific utilities in favor of using generally available functions (such as sine and cosine functions) and straightforward "C" code. As previously mentioned, the final image is stored in the Utah.rle format. The use of this format was justified because it is a public domain format that is widely supported for many different systems. Although developed on a DEC GPX Microvax III system, I have been able to transfer, recompile, and run VIPER on a Silicon Graphics 3130 Iris system as well as a VAX 11/785 computer.

4.2 Suggested Enhancements

VIPER is not intended as an ultimate graphics tool for engineers. It is also not intended to be used for quantitative measurements. However, I do believe it has the potential to become a very powerful engineering tool by providing 3D images of aerodynamic flow field data. These images will make qualitative assessment of flow fields relatively simple and intuitive for engineers. In the course of this thesis, I have developed a prototype for VIPER that is simple to use and provides many of the features an engineer is likely to require or desire. Although useful in its current form, there are several features that could be incorporated into VIPER to make it even more robust and powerful.

4.2.1 Grid Configuration. VIPER operates on the assumption that the data volume will be represented by a rectangular grid. It was not until the later stages of my thesis work that I discovered that aerodynamicists rarely use a rectangular grid; instead, they routinely use a grid that is nearly orthogonal but not rectangular (see Figure 13). To compensate for this difference in grids, I would suggest developing an intermediary program that could map a rectangular grid into a given non-rectangular grid, trilinearly interpolate the data value at that point, and produce a new data file that contains a rectangular approximation of the original flow field grid. Although this would provide a temporary bridge between the different grid configurations, it might not be a good long-term solution. To begin with, it imposes an additional burden upon the user. Secondly, using interpolation to develop a rectangular grid and then interpolating again to locate target surfaces implies the potential loss of accuracy in the course of the rendering process. Although VIPER is intended as a qualitative tool and can therefore tolerate *some* inaccuracy, it is clearly desirable to limit this loss of precision as much as possible.

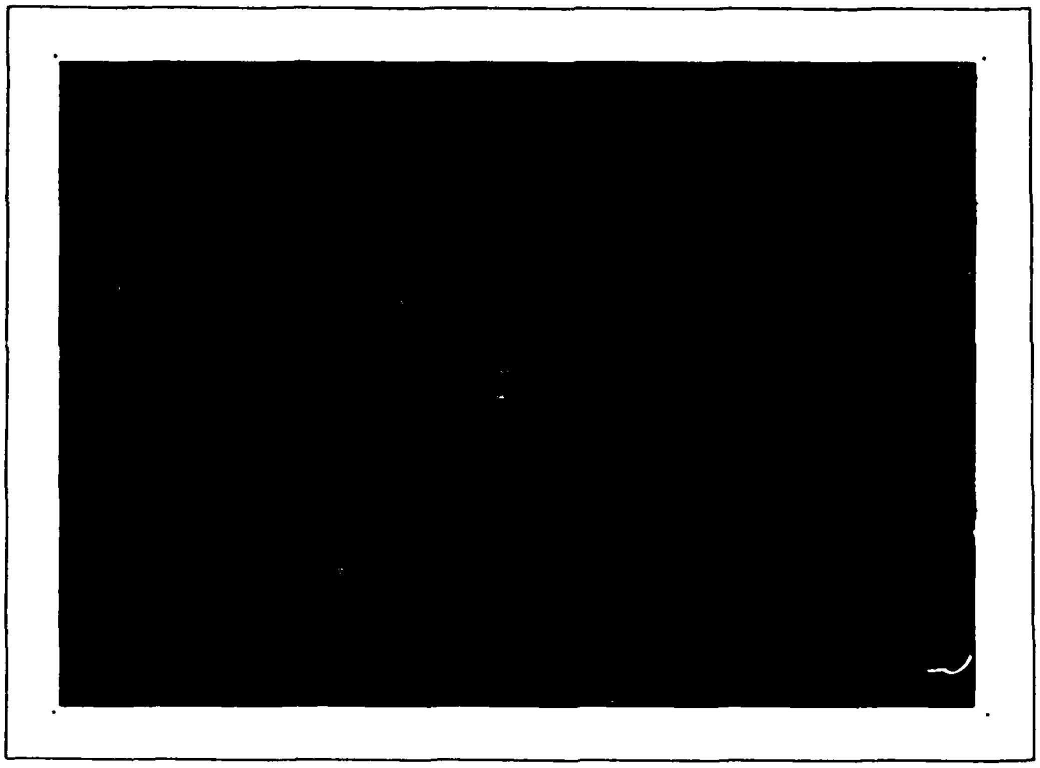

Figure 12. Subvolume Isolation Via Slab Removal Method.

VIPER can now process irregularly-sized voxels; I don't believe non-rectangular voxels will require any major changes to the trilinear interpolation or volume traversal routines. However, I don't see any alternatives to storing the X,Y,Z coordinates for each mesh node. If this proves necessary, the memory required to store or represent the data file will be quadrupled; this may make

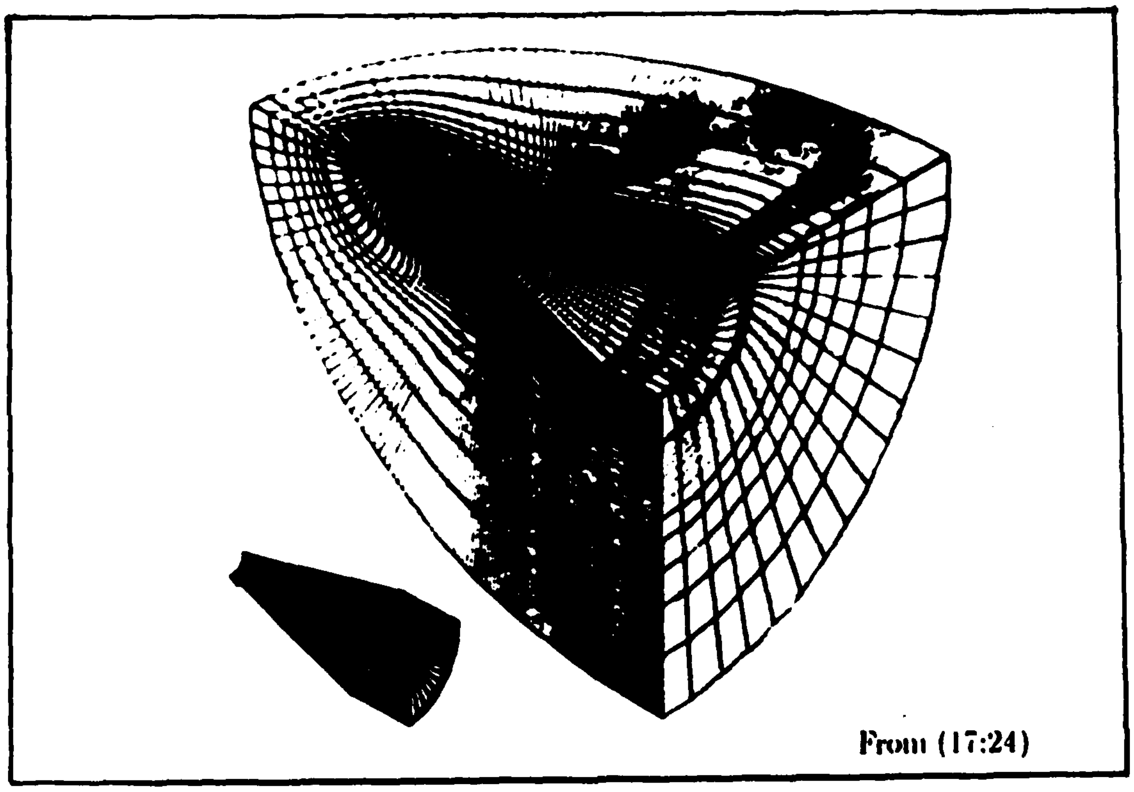

Figure 13. Typical Aerodynamicist's Grids

VIPER impractical for use on many systems. Therefore, I would suggest this problem be explored in an effort to produce a procedure that would allow a generalized description of the transform required to map grids from non-rectangular to rectangular forms.

4.2.2 Non-Scalar Values. Although 60 per cent of the standard flow field variables are vectors components, VIPER can only display scalar data. Vector data *may* be displayed; however, since only one vector component could be displayed per image, the value of such images is highly questionable. Therefore, I recommend the problem of generating useful displays of vector data be examined. One possibility would be to combine the three velocity vectors (V_r, V_θ, V_z) and displaying the magnitude of the resultant vector as a scalar value.

4.2.3 Parameter Derivation. Closely related to the problem of displaying vector values is the need to display data that is derived from the standard flow field values of pressure, energy, and velocity vectors. In order to provide this capability, a language processor must be developed so that the user may describe how the desired parameter is derived. My feeling is that this feature should be developed as a separate program rather than as an integral part of VIPER; otherwise, VIPER may become too cumbersome to use.

4.2.4 Supercomputers and Parallel Processing. As previously mentioned, VIPER takes too long to generate images to be used for interactive viewing. However, it is possible that if installed on a supercomputer or broken up into several parallel processes, the time required to complete an image could possibly be reduced to seconds instead of hours. Restructuring VIPER for parallel processing should be relatively simple: individual segments of the viewing plane (*e.g.*, a few scan lines) could be assigned to different processors. When finished, the segments would be reunited to form the final image. The potential for improved performance definitely makes this area worth exploring.

Should VIPER's execution time be shortened enough to permit interactive viewing, new user interfaces will need to be developed. These interfaces will have to allow the user to rotate the volume, change the viewpoint, and isolate various subregions of the volume. It would also be useful to allow the user to select different target values or even change the type of data being displayed (*e.g.*, from velocity to pressure) "on the fly".

4.2.5 Automatic Flow Field Feature Detection. One of VIPER's shortcomings is that the user must have some idea of what data values will be of interest. If the user doesn't already have some idea of what the flow field should look like, he must rely on chance to find such features as shock waves and vortices. One way to locate these features is to preprocess the data to find minimum, maximum , and average values. With this information, the user could render an image using target values and wide enough windows to span all of the data. With successive images, critical flow field features may be ferreted out. This is clearly an awkward, brute force approach to an important issue. It would be better if VIPER could automatically locate and render areas of potential interest to engineers. This feature might take the form of having the user specify the parameter to be monitored and a threshold for the local gradient. If VIPER detected a local gradient that exceeded this threshold, it would render the region and save the parameter values in a separate file for post-rendering analysis. Such a feature could save a great deal of time and frustration for the user.

4.2.6 Temporal Features. Many flow field features such as shock waves and vortices develop and change as a function of time. It would therefore be useful to be able to specify a start and stop time along with the number of images to be produced and have VIPER automatically develop the desired sequence of images. This capability would obviously require the user to supply some means of associating a specific time for each data sample. If this could be done in conjunction with supercomputing and parallel processing (see above), the possibility of automatic "movie" generation and interactive image viewing might become very real indeed.

4.2.7 Subvolume Isolation. VIPER allows the user to isolate data subvolumes by explicitly removing layers from the volume faces (which is much like peeling the layers off of an onion) and embedding the viewing plane within the data volume (which is like making one slice in an onion). However, users will often want to remove sections which do not lie parallel to the volume faces. Instead, they will need the ability to carve out any arbitrary subvolume they so desire. This capability should be provided.

4.3 Possible Applications

Maxine Brown recently observed that there aren't very many visualization tools that run on workstations. "Each scientist is investigating some little aspect of some far-fetched problem, so they are not going to find general tools" she notes. "In fact, that is a problem; there is a level where they all need similar tools, but then past that level, they all need a great degree of specialization (25:25)." I have designed VIPER to operate on a variety of workstations and systems. I also developed VIPER in a general manner so that specialized applications and general enhancements could be easily incorporated. Although the primary objective of this thesis was to produce a program that could generate images of aerodynamic flow field data, volumetric rendering techniques lend themselves to a wide variety of possible uses. In the course of this thesis, I have rendered images of aerodynamic flow fields, error levels in magnetic fields, and atomic energy levels between crystals without modifying VIPER. This leads me to conclude that VIPER qualifies as a "general tool". I also believe that, if necessary, VIPER could be easily modified to accomodate specialization in various fields, some of which are discussed below.

4.3.1 Radar Cross Sections. Radar cross sections are an important consideration when designing aircraft and missiles, especially at a time where emphasis on "stealth technology" is increasing. Since radar cross section data is collected in a 3D mesh, VIPER could be used to assess radar returns from many look angles as well as different radio frequencies and power settings.

4.3.2 Threat Envelopes. Whether emplaced around an area or point target, radar coverage is affected by the target's surrounding terrain. Also, there may be zones of little or no radar coverage between installations. Likewise, anti-air defenses (SAM and AAA) are susceptible to the same terrain masking and site separation considerations. The air combat arena is somewhat unique in that it is three-dimensional instead of the two-dimension realm associated with ground or naval surface combat. Therefore, it should be possible to model the volume around a given target to produce an image that identifies potentially vulnerable as well as high-threat regions. VIPER could be used in such a scenario as an aid to mission planning and risk assessment. VIPER could also be used as an intelligence tool if used to analyze key signature elements or perform strength/weakness assessments of weapons systems such as enemy fighters, naval surface combatants, or tanks.

4.3.3 Earth Sciences. Volumetric rendering techniques have already been applied to meteorological data. Several possible uses for VIPER may also be found in the earth sciences. One area of research that could make use of VIPER involves the study of isothermic layers in the ocean. Although generally understood, there is still no way of predicting their thickness or extent. These layers provide excellent concealment for submarines; if the ocean surrounding a surface combatant could be sampled in several places and the resultant data was processed by VIPER, a 3D image of the isothermic layers in the immediate region could be produced; this would allow naval commanders to narrow down areas where enemy submarines could be hiding. It would also allow detection equipment operators to fine-tune their equipment to compensate for a given isothermic configuration.

Another earth science application for VIPER could be found in the realm of seismic data. Information from core samples and seismic sensors could be integrated and rendered to yield a 3D image of various features. Applications would include such diverse activities as earthquake prediction, petroleum exploration, and assessing the effectiveness of hardening techniques around missile silos, bunkers, and other underground structures.

4.4 Summary

Rendering images based on volumetric data is a relatively new field of endeavor in computer graphics. For this thesis, I have developed VIPER, a prototype volumetric rendering program. After producing 3D images of aerodynamic flow field, atomic energy levels, and magnetic sensor data, I am convinced that VIPER may be used as a powerful engineering tool. The comments received from several potential users seem to confirm this belief and justify more development effort in this area. Although volumetric rendering cannot possibly replace other imagery tools such as interactive particle traces, I feel that it is destined to take its place as an augmentation to the array of aids available to aerodynamicists in their pursuit of safe, efficient aircraft design. While developing VIPER and analyzing its applicability to aerodynamic engineering, I have become aware of the range of possible uses for programs such as VIPER. It is my belief that as each area is explored, more uses will be continuously uncovered. Therefore, I believe that VIPER should be enhanced and its applicability to aerodynamic flow fields as well as other areas of research and development should be pursued.

Appendix A. *User's Guide*

A.1 Introduction

The Volumetric Imaging Program for Engineering Research (VIPER) is a general purpose graphics program designed to produce three-dimension (3D) images based on meshes of volumetric scalar data. VIPER is semi-interactive in that it allows the user to interactively set the parameters that control the image to be produced.

This User's Guide will outline how to install VIPER on a typical computer system and how to determine the parameter values you will need to produce the image *you* want. Finally, some hints for speeding up image production will be provided.

Although VIPER was developed on a Unix operating system, I have tried to keep the program as system-independent as possible. Therefore, I believe that VIPER should run on any system that supports the 'C' programming language and has sufficient memory to handle large data files. Likewise, this User's Guide is biased towards Unix systems; however, outside of some minor installation details, there should be no real complications involved with using VIPER on non-Unix systems.

A.2 Installing VIPER

VIPER actually requires two libraries in order to operate. The first library is composed of the files that make up the Utah.rle image format routines. These routines are a public domain set of files that enable VIPER to produce more efficient image files as an output. The second VIPER library is comprised of the files that produce VIPER itself.

A.2.1 Utah.rle Library Table 1 lists the files that make up the Utah.rle library. Some of these files may need to be modified to operate on your computer system. Instructions for modifications may be found within the files and are beyond the scope of this user's guide. Once these files have been installed, you will need to compile and link them to form the final library module **librle.a**. The simplest way to do this is to rename **rlemakefile** to **makefile** and then enter the command *make*. Doing this will cause **makefile** to automatically produce **librle.a** for you (*Note:* This applies to Unix systems; if you are a non-Unix user, **makefile** may not work for you. Consult your system manager or documentation to determine the compile/link operations you will need to perform for this step.). Once **librle.a** has been prepared, you are ready to install VIPER.

bstring.c	rle_getrow.c	rlemakefile	sv_putraw.c.fi
buildmap.c	rle_getrow.c.b	Runsv.c	sv_putrow.c
dither.c	rle_getrow.c.f	scanargs.c	svfb_global.c
rle_getcom.c	rle_putcom.c	svfb.h	svfb_global.h
rle_getraw.c	rle_raw_alc.c	sv_putraw.c	XtndRunsv.h
rle_getraw.h	rle_row_alc.c	sv_putraw.c.ba	

Table 1. Utah.rle Library Files

A.2.2 VIPER Library The files you will need for VIPER are shown in Table 2. Once these files are in place on your system and you have produced a copy of **librle.a** (see above), you are ready to build the VIPER library. To do this, rename **vipermakefile** to **makefile** and enter the command *make viper* to complete the VIPER-building sequence (again, non-Unix users may need to follow a different procedure.). You should now have all you need to produce 3D images of your volumetric data.

A.3 Using VIPER

Using VIPER is a fairly straightforward process that requires a file of properly formatted scalar data and a few user-supplied parameter values.

color_ops.c	misc.c	vglobals.h
externs.h	setup.c	viper.c
interpolate.c	traversal.c	viper.h
mapval.c	vexterns.h	vipermakefile
matrix_ops.c		

Table 2. VIPER Library Files

A.3.1 Data Format Figure 14 shows a small file of data in the VIPER format. The first line is simply a remarks line. This is a line of up to 80 characters and is reserved for any labels or pertinent information that identifies the file. The contents of this line are immaterial to VIPER; however, **this line must be present, even if it is blank**. The second line contains three integers that indicate the dimensions (in terms of data nodes) along the X, Y, and Z axes of the data volume to be rendered. The remaining lines are comprised of the X, Y, Z coordinates of each data point along with the scalar data value at that point (*Note:* Putting one set of coordinates and value per line makes the file easier to read; however, the file will require more memory space if you do this. For larger files, it may be better to record more than one data point per line. VIPER can read either format.). Remember, **VIPER assumes that you are using a *rectangular* grid of data points!** Any other type of grid will be distorted and the final image will be difficult (if not impossible) to interpret.

```
Sample VIPER Data File - Comment Line
2     2     2
-2.0  -2.0  0.0  -18.000
-2.0  -2.0  1.0  12.000
-2.0  -1.0  1.0  6.000
-2.0  -1.0  2.0  8.000
-1.0  -2.0  1.0  6.000
-1.0  -2.0  2.0  6.000
-1.0  -1.0  0.0  0.000
-1.0  -1.0  2.0  3.000
```

Figure 14. Sample VIPER Data File

A.3.2 Parameter Setting VIPER will require you to set several viewing parameters. In order for you to be able to make informed value selections, we will now review each parameter and how to determine the values you will need.

A.3.2.1 Command Line Entries. The first parameters you will set are embedded in the command line used to invoke VIPER. The command line format is

```
viper -option input output
```

The **-option** allows you to locate the position of the image. Using **-c** will center the image on your screen while **-k** will locate the image in the lower left corner. Normally, you will probably want to center the image; the **-k** option was developed for special situations (such as producing videotapes of images).

The **input** parameter is the name of the data file you wish to have rendered; **output** is the name you assign to the final image file (*Note:* This file name must have a **.rle** extension.).

If your command line doesn't have the proper format, VIPER will reject it and terminate. Otherwise, VIPER will read your data file and then enter the interactive parameter-setting phase.

A.3.2.2 Feedback. The first value you will be asked for will determine whether or not VIPER produces feedback for you during image generation. This feedback takes the form of echoing back some of your parameter selections as well as printing an advisory notice after each scanline in your image has been completed. This information is useful if you want to monitor the progress of your image as it progresses as well as the first few times you use VIPER. Once you get used to running VIPER or want to put VIPER into the background mode, you will probably want to disable the feedback option. To enable feedback, enter **1**; to disable feedback, enter **0**.

4.6.2.6 *Scaling.* After VIPER has read your data file, it will let you know the final volume dimensions and ask you for a scaling factor. This is opportunity to change the physical dimensions of your volume along any or all of the volume's axes. However, to determine the proper value will require you to set several parameters. In order to work out these values, we will now look at the viewing geometry that VIPER uses (see Figure 15). VIPER assumes that you are at the axes origin and looking straight down the Z axis. The viewing plane and data volume are initially centered on the Z axis. VIPER will allow you to set the distances from your eye to the viewing plane as well as the center of the data volume. This capability allows you to use VIPER in much the same way as you would use a camera. Moving the viewing plane is much like adjusting a camera's focal length while moving the volume corresponds to moving towards or away from an object in order to adjust the total area it will occupy in a photograph. However, VIPER gives you one more option: placing the viewing plane *inside* the volume allows you to cut away portions of the volume: whatever is between your eye and the viewing plane will be omitted from the final image.

Referring to Figure 16, we see how to determine the necessary parameter settings by using similar triangles. We already know the size of the data volume, d. If we then choose a value for the distance from our eye to the viewing plane (a), the desired extent of the volume in the final image (b), and the distance from our eye to the *center* of the data volume c, we may easily find the scaling factor required to produce this image by the following equation:

$$Scale\ Factor = \frac{b \times c}{a \times d}$$

Once you have determined the scaling factor, you should enter it into VIPER. If you do not enter the same value for all three axes, the data volume will be "stretched" from its original proportions. Another thing to keep in mind is that if you don't want to change the volume's size, you should enter a value of 1.0.

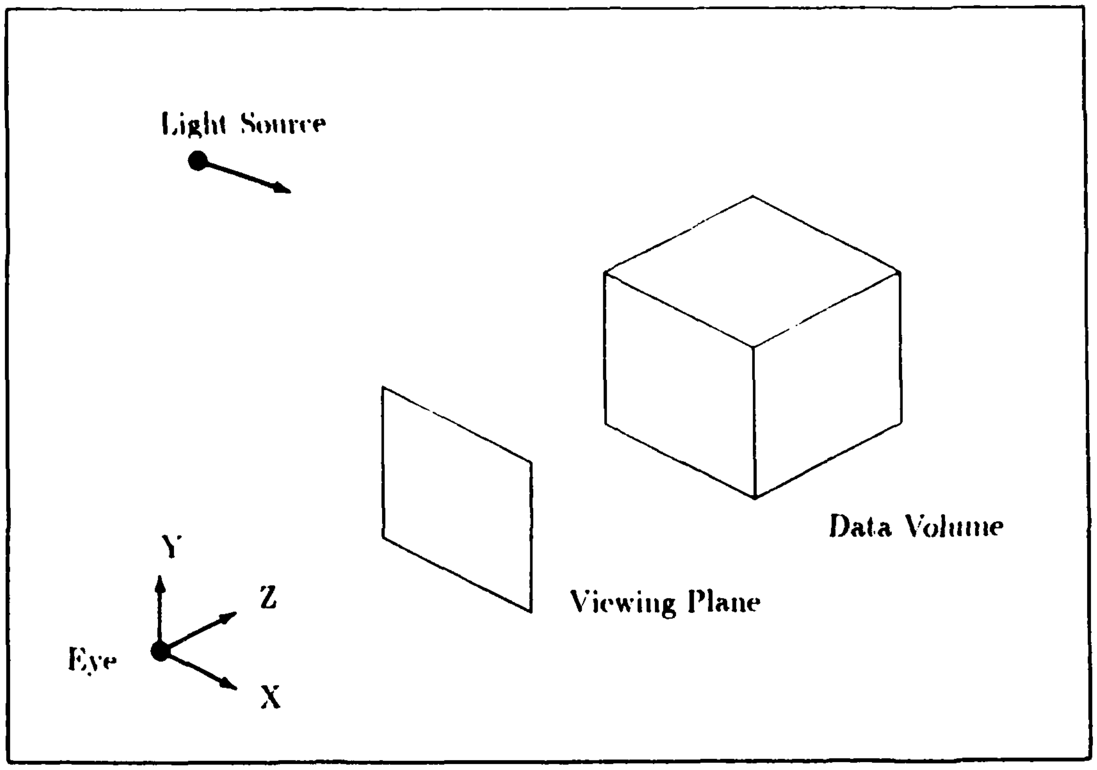

Figure 15. VIPER's Viewing Geometry

A.3.2.4 Volume Trimming. As previously noted, you can use the viewing plane to eliminate portions of the data volume. VIPER offers you another method for trimming off portions of the volume. You will be asked if you want to trim "slabs" off of the volume's faces. To decline this opportunity, simply enter **0**. However, if you do want to trim away part of the volume, entering **1** will cause VIPER to prompt you for decisions regarding the six volume faces. Enter **0** for faces you don't want to trim and **1** for faces you do want to trim. VIPER will ask you how much of the volume you want to trim off; simply enter the amount you want removed (Remember your scaling factor when you enter this value!).

A.3.2.5 Object Distances. The next values VIPER will ask for are the distances from your eye to the viewing plane and from your eye to the *center* of the data volume, respectively.

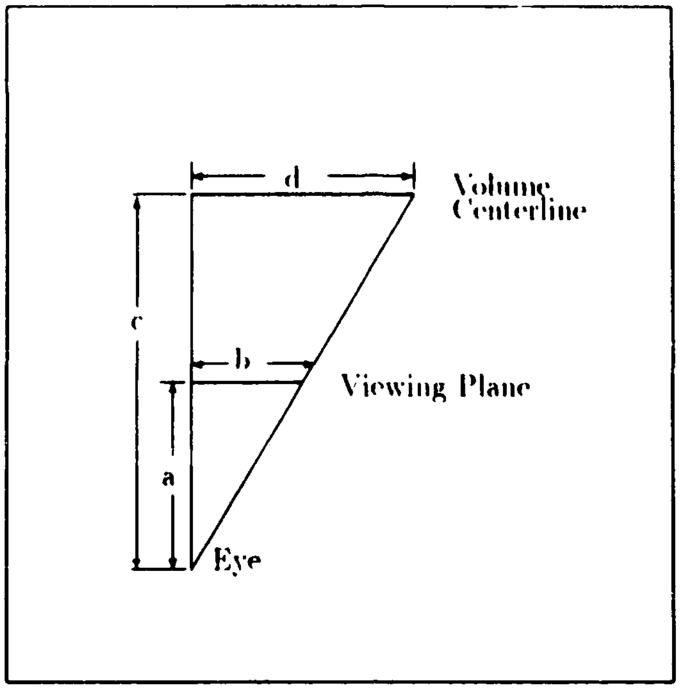

Figure 16. Calculating Object Positions and Scaling Factors

A.3.2.6 Manipulating the Data Volume. VIPER will next give you the opportunity to reorient the data volume. The first set of parameters will allow you to rotate the volume about any or all of its three axes. These values are measured in degrees and increase in a anti-clockwise direction (as you look along the axes). The second set of parameters permit you to "slide" the volume along the three axes. To move left, down or towards you, you will need to enter a negative value; likewise, positive values will move the volume right, up, and away from you.

A.3.2.7 Changing the Viewing Plane Dimensions. The viewing plane is measured in terms of picture elements (*pixels*). The default horizontal and vertical dimensions for the viewing plane are 1024 × 768, respectively. To maintain these values, enter a **0** when VIPER asks you if you want to change the screen size. To change these dimensions, enter a **1** followed by the new horizontal and vertical dimensions you want (*Note:* If the values you enter exceed the default values or are less than two, VIPER will revert to the default screen size.).

A.3.2.8 Lights. The next set of parameters will determine the lighting within your scene. The first value represents the ambient light present. This is the amount of light that would be available if none of your light sources were turned on. A value of 0.0 corresponds to a totally darkened room while entering 1.0 would indicate a bright, sunlit scene. Next, VIPER will ask you how many light sources you want to install. You must have at least one light but no more than five lights. For each light source, you will need to supply VIPER with the three coordinates for the light's location (Remember, your eye is at the origin with a location of [0,0,0]. A light 100 units directly above your eye would have coordinates [0,0,100].). When all of your lights have been activated, you will need to tell VIPER what it is supposed to be looking for.

A.3.2.9 Setting the Target Values. VIPER will try to find surfaces whose values correspond to the targets you set for it. You may specify from one to five target values to be sought. After you tell VIPER how many values are to be found, you will need to provide a set of values for each target you select. The first parameter is the value of the surface itself. You may then specify a "window" of values that are "close enough" to the target value (*e.g.*, if you specify a target value of 50.0 and an allowable variance if 5.0, the window of values will range from 45.0 to 55.0).

The next parameters will determine the surface characteristics of color and opacity. VIPER produces color by additively blending red, green, and blue (RGB) to yield the desired color (*Note: additively* means that lower color values are darker; thus, an RGB value of 000 produces black; 001 is blue, 111 is white, etc). For each surface, you will need to provide an RGB triplet to specify the color of the surface. Be careful not to give the same RGB value to more than one target; both will be given the same color and you won't be able to tell the two apart. Finally, you will provide an opacity coefficient for the surface. This will determine how transparent the surface will be. A value of 0.0 will be completely transparent and won't show up in the image; an opacity of 1.0 will be completely opaque and thus hide any underlying surfaces. Any value between these two limits will provide a partially-transparent surface.

A.3.2.10 Surface Reflectivity. There is one final surface parameter that is applied uniformly to all surfaces. The surface reflectivity of the surfaces ranges from 0.0 for a very dull surface to 1.0 for a very shiny surface. You will be able to determine the glossiness of the surface by entering a single value within these two bounds.

A.3.2.11 Image Matte. The final parameter you will be prompted for is the RGB triplet for the image matte. This is a default color that will be applied to any part of the viewing plane that is not occupied by the data volume. The image matte is useful in a number of ways. To begin with, the matte will allow you to see where the data volume is. This not only helps you to fine tune your viewing geometry in subsequent images, but also helps you to orient rendered surfaces in relation to the volume. If you don't want an image matte, simply enter the RGB triplet for black (0,0,0).

Once you have entered this sequence of parameter values, VIPER will enter its autonomous mode and proceed with generating the image you have specified. When VIPER has completed your image, it will ask you if you want to generate another image. Entering **0** will cause VIPER to terminate. On the other hand, if you enter **1**, VIPER will retain the data volume and scaling factors from the original image and reenter the parameter-setting phase. This cycle may be repeated for as many images as you desire. Thus, you can save a great deal of time by not having to repeat the data-reading portion of the program.

4.7 *Speeding Up Image Generation*

There are a number of ways to shorten the time required to generate an image with VIPER. Most of them entail some sort of tradeoff between the time required and the detail and quality of the final image. This section will briefly outline some of these techniques and any possible tradeoffs that I have been able to identify.

The first technique involves cutting down the size of the viewing plane as much as possible. Since the time required to generate an image is directly proportional to the number of pixels that comprise the image, cutting down on the number of pixels will cut down on the time required. This is a fairly good technique in that it doesn't necessarily degrade the final image. Instead, it requires you to carefully size your image of the volume and trim the viewing plane accordingly. Naturally, if you use a smaller image size for your volume, you will be able to trim the viewing plane even more. Also, smaller image sizes will produce a faster run, even if you don't trim the screen.

Another way to shorten the execution time is to trim away unnecessary portions of the data volume. As previously mentioned, this can be accomplished either by embedding the viewing plane within the data volume or explicitly defining slabs of the volume that are to be trimmed off. This speeds up the rendering process by presenting less data to be rendered. You should be very careful, however, when using this technique. Indiscriminate trimming may cause VIPER to omit important features within the volume.

Using very dull, opaque surfaces will produce a shorter run time. However, opaque surfaces will mask any underlying surfaces. However, this technique is useful for "quick looks" and situations where you know there will be no underlying surfaces.

A final technique I have used to speed up the image time is to place the viewing plane very close to the eye position. This will cause the rays used for image rendering to be more divergent. As this divergence increases, the time required to produce an image decreases. At the same time, more divergent rays produce ever-increasing gaps where no data is included in the final image. Significant features (such as vortices and shock waves) could be concealed in these gaps. Once again, this technique is more suited for "quick look" images or images of volumes that contain only coarse features.

As I previously mentioned, these techniques will speed up the rate of image execution; however, you must be careful and weigh the consequences involved before proceeding with these or any other shortcuts.

A.5 *Sample Session*

The following is a transcript from an actual VIPER session. The boldfaced entries are VIPER prompts; user inputs are shown in standard print.

```
viper -c error.vol
Do you want feedback (0 = NO, 1 = YES) ⇒0
Polhemus Error Field
Please enter name for image file ⇒sample1.rle
The Volume Being Rendered Has XYZ Voxel Dimensions of 5 5 3.
and Goes From -2.00 to 2.00 Along the X Axis,
-2.00 to 2.00 Along the Y Axis,
and -1.00 to 1.00 Along the Z Axis.
Enter X, Y, & Z Scaling Factors ⇒15 15 15
Want to Trim Away Part of the Volume? (No:0 Yes:1) ⇒1
Trim Off the LEFT Side? (No: 0 Yes: 1) ⇒0
Trim Off the RIGHT Side? (No: 0 Yes: 1) ⇒0
Trim Off the TOP? (No: 0 Yes: 1) ⇒0
Trim Off the BOTTOM? (No: 0 Yes: 1) ⇒0
Trim Off the FRONT? (No: 0 Yes: 1) ⇒1
How Much? ⇒5.7
Trim Off the BACK? (No: 0 Yes: 1) ⇒0
How Far From Your Eye to CENTER OF VOLUME ⇒1500.
How Far From Your Eye to the Viewing Plane ⇒1000.
Enter Angle of Rotation About X Axis (Degrees) ⇒0.
Enter Angle of Rotation About Y Axis (Degrees) ⇒35.
Enter Angle of Rotation About Z Axis (Degrees) ⇒0.
Enter X, Y, and Z Translation Distances ⇒0. 0. 0.
Do you want to change the screen size (0 = NO, 1 = YES) ⇒1
Enter new screen dimensions (x y) ⇒350 350
Enter Ambient Light Level (0.0 → 1.0) ⇒.8
How Many Light Sources Do You Want ⇒2
Enter X, Y, Z Location for Light # 1 ⇒35. 50. 200.
Enter X, Y, Z Location for Light # 2 ⇒-35. 200. 800.
Enter number of target values to be found ⇒2
Enter target value # 1 ⇒0.
Enter allowable target value variance ⇒.5
Enter RGB values to be associated with 0.00 ⇒ .5 0.
Enter Opacity for 0.00 (Clear:0.0 → Opaque:1.0) ⇒.8
Enter target value # 2 ⇒2.
Enter allowable target value variance ⇒0.75
Enter RGB values to be associated with 2.00 ⇒0. 0. 1.
Enter Opacity for 2.00 (Clear:0.0 → Opaque:1.0) ⇒1.
Enter Surface Glossiness (Dull: 0.0 → Shiny: 1.0) ⇒ .95
Enter RGB Values for Border Matte ⇒ 1. 0. 0.
Saving File as sample1.rle
One more time? ( 0 ⇒ No, 1 ⇒ Yes ): 0
```

4.6 *Conclusion*

In developing VIPER, I have tried to produce a convenient, reasonably efficient program that will render 3D images of volumetric data. I have tried to incorporate as many useful features as I could. If you can think of any other capabilities that would be useful (or implement any of them yourself) or encounter problems that I have missed, please notify the computer graphics faculty here at the Department of Electrical Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio.

Bibliography

1. Amburn, Capt E. P. *The Graphical Display of Multi Dimensional Aerodynamic Flow Field Data.* MS thesis, AFIT/GCS/MA/79D-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1979.

2. Beran, Capt P. Personal interview. Air Force Institute of Technology, Wright-Patterson AFB OH, 25 August 1988.

3. Boehm, W. "The de Boor Algorithm for Triangular Splines", *Surfaces in Computer Aided Geometric Design*, edited by R. E. Barnhill and W. Boehm. Amsterdam: North-Holland Publishing Company, 1983.

4. Combs, Harry. *Kill Devil Hill.* Boston: Houghton Mifflin Company, 1979.

5. Drebin, R. A., *et al.* "Volume Rendering", *Computer Graphics 22:* 65 - 74 (August 1988).

6. Finley, Dave. "Researchers Use CRAY-2 for Aerodynamic Simulations", *Vector View 1:* 1,8 (May 1988).

7. Foley, J. and Van Dam, A. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley Publishing Co. Reading, MA: 1982.

8. Frieder, G. *et al.* "Back-to-Front Display of Voxel-Based Objects", *Computer Graphics and Applications* 5:52 - 60 (January 1985).

9. Fujimoto, A. *et al.* "ARTS: Accelerated Ray-Tracing System", *IEEE Computer Graphics and Applications 6:* 16 - 26 (April, 1986).

10. Glassner, A. "An Overview of Ray Tracing", *Siggraph '87 Tutorial on Ray Tracing:*1 - 36.

11. Glassner, A. "Space Subdivision for Fast Ray Tracing", *IEEE Computer Graphics and Applications 4*:15 - 22 (October, 1984).

12. Haber, R. "Visualization in Engineering Mechanics: Techniques, Systems and Issues", *Visualization Techniques in the Physical Sciences", SIGGRAPH '88*:89-118.

13. Karamcheti, K. *Principles of Ideal-Fluid Aerodynamics.* Malabar, FL: Robert E. Krieger Publishing Co. 1966.

14. Kroos, Kenneth A. "Computer Graphics Techniques for Three- Dimensional Flow Visualization",*Frontiers in Computer Graphics*, edited by Tosiyasu L. Kunii. Tokyo: Springer-Verlag, 1985.

15. Levoy, Marc. "Display of Surfaces from Volume Data", *IEEE Computer Graphics and Applications*, 8:29-37 (May 1988).

16. Lorensen, W. E. and Cline, H. E. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", *Computer Graphics, 21*:163 - 169 (July 1987).

17. NASA Langley Research Center, *Computational Fluid Dynamics*, Hampton, VA (Undated).

18. McCormick, B. H. *et al.* "Visualization in Scientific Computing", *Computer Graphics, 21:* A13 - A15 (November 1987).

19. Nielson, G. M. and Franke, R. "Surface Construction Based Upon Triangulations", *Surfaces in Computer Aided Geometric Design*, edited by R. E. Barnhill and W. Boehm. Amsterdam: North-Holland Publishing Company, 1983.

20. Sandor, J. "Octree Data Structures and Perspective Imagery", *Computers and Graphics 9:* 393 - 405 (1986).

21. Shang, J. "Current Topics in Computational Fluid Dynamics", Address to AFIT students. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 31 March 1988.

22. Smith, R. *et al.* "Visualization of Computer-Generated Flow Fields", *Flow Visualization, Volume 3*, edited by W. J. Yang. Washington, D. C.: Hemisphere Publishing Company, 1985.

23. Whitted, T. "An Improved Illumination Model for Shaded Display", *Communications of the ACM 23:* 343 - 349 (June 1980).

24. Winkelmann, A. E. and Tsao C. P. "Flow Visualization Using Computer-Generated Color Video Displays of Flow Field Survey Data", *Flow Visualization, Volume 3*, edited by W. J. Yang. Washington, D.C.: Hemisphere Publishing Company, 1985.

25. Wolfe, A. *et al* "The Visualization Roundtable", *Computers in Physics:*16 - 26 (May/June 1988).

26. Wolff, R. "Visualization in the Eye of the Scientist", *Computers in Physics:* 28 - 35 (May/June 1988).

Vita

Captain David James Bridges was born [redacted] [redacted] He graduated from Marion A. Peterson High School in [redacted] After marrying [redacted] and the birth of [redacted] daughter, [redacted] Dave enlisted in the United States Air Force in 1976. Upon completion of Air Force Basic Training at Lackland Air Force Base (AFB), Texas, Airman First Class Bridges attended Imagery Interpretation training at Lowry AFB, Colorado. His first post-training assignment was to the 544th Aerospace Reconnaissance Wing, at Offutt AFB, Nebraska. While there, Dave celebrated the birth [redacted] His next assignment was at Kadena Air Base, Okinawa, Japan, where Staff Sergeant Bridges was a shift supervisor in Detachment 1 of the 9th Strategic Reconnaissance Wing. In 1981, Dave was accepted into the Airman Education and Commissioning Program (AECP) and attended Colorado State University in Ft. Collins, Colorado. After receiving his Bachelor of Science degree in computer science, he attended Officer Training School in San Antonio, Texas, and was subsequently assigned to the 57th Fighter Weapons Wing, Nellis AFB, Nevada as the configuration manager for all Tactical Air Force F-111D/F and A-10 operational flight programs. In 1986, Lieutenant Bridges was selected to attend the Air Force Institute of Technology in pursuit of a Master's degree in computer systems.

[redacted]

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/88D-2	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering; 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFWL; 8b. OFFICE SYMBOL (If applicable) SCP	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Kirtland AFB, NM 87117	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
See box 19

12. PERSONAL AUTHOR(S)
David J. Bridges, B.S., Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS thesis	FROM ______ TO ______	1988 December	71

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
20	01	
12	05	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Flow Fields
Computer Graphics

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: VOLUMETRIC RENDERING TECHNIQUES FOR THE DISPLAY OF THREE-DIMENSIONAL AERODYNAMIC FLOW FIELD DATA (UNCLASSIFIED

Thesis Advisor: Phil Amburn, Major, USAF
Professor of Computer Systems

[illegible]
12 Jan 1989

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED [✓ SAME AS RPT.] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Phil Amburn, Major, USAF	22b. TELEPHONE (Include Area Code) (513) 255-3576; 22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ABSTRACT

The behavior of fluids has been studied for years. This behavior has been modeled by the Navier-Stokes equations since the mid-nineteenth century; however, these equations are so complex that their use by engineers was impractical until the advent of the modern computer. The same computers that made these equations useful [illegible] problem was created; a new problem: data saturation. The solution to the Navier-Stokes equations, which represent an aerodynamic flow field ~~is~~ comprised of thousands ~~if not~~ millions of numbers. This much data makes it virtually impossible to conceptualize the situation within the flow field. To alleviate this problem, various computer graphics techniques have been used to provide images of flow fields. Some of these techniques rely on approximating isometric surfaces of interest with polygons; others only provide an image of a two-dimensional "slice" of the flow field.

This thesis ~~project~~ applies a relatively new graphics technique known as "volumetric rendering" to generate three-dimensional images of aerodynamic flow fields. Volumetric rendering has shown promising results in several applications of scientific visualization. One advantage of this technique ~~is~~ that it doesn't rely on geometric primitives to approximate a surface; Also, no *a priori* knowledge of the flow field's form is required to produce the final ~~image~~.

To assess the utility of volumetric rendering in the field of computational fluid dynamics (CFD), a program known as VIPER is developed. This thesis outlines the requirements and specific algorithms for this program. Images of flow fields are presented and discussed along with program enhancements. Although VIPER is developed for CFD purposes, its design permits its use in ~~different~~ applications, several of which are discussed.